1. 磁盘使用率检测（用shell脚本）
2. LVS 负载均衡有哪些策略?
3. 谈谈你对LVS的理解?
4. 负载均衡的原理是什么?
5. LVS由哪两部分组成的?
6. 与lvs相关的术语有哪些?
7. LVS-NAT模式的原理
8. LVS-NAT模型的特性
9. LVS-DR模式原理
10. LVS-DR模型的特性
11. LVS三种负载均衡模式的比较
12. LVS的负载调度算法
13. LVS与nginx的区别
14. 负载均衡的作用有哪些?
15. nginx实现负载均衡的分发策略
16. keepalived 是什么?
17. 你是如何理解VRRP协议的
18. keepalived的工作原理?
19. 出现脑裂的原因
20. 如何解决keepalived脑裂问题?
21. zabbix如何监控脑裂?
22. nginx做负载均衡实现的策略有哪些
23. nginx做负载均衡用到哪些模块
24. 负载均衡有哪些实现方式
25. nginx如何实现四层负载?
26. 你知道的web服务有哪些?
27. 为什么要用nginx
28 . nginx的性能为什么比apache高?
29 . epoll的组成
30 . nginx和apache的区别
31. Tomcat作为web的优缺点?
32. tomcat的三个端口及作用
33. fastcgi 和cgi的区别
34. nginx常用的命令
35. 什么是反向代理，什么是正向代理，以及区别?
36. Squid、Varinsh、Nginx 有什么区别?
37. nginx是如何处理http请求的
38. nginx虚拟主机有哪些?
39. nginx怎么实现后端服务的健康检查
40. apache中的Worker 和 Prefork 之间的区别是什么?
41. Tomcat缺省端口是多少，怎么修改
42. Tomcat的工作模式是什么?
43. Web请求在Tomcat请求中的请求流程是怎么样的?
44. 怎么监控Tomcat的内存使用情况
45. nginx的优化你都做过哪些?
46. Tomcat你做过哪些优化
47. nginx的session不同步怎么办
48. nginx的常用模块有哪些?
49. nginx常用状态码
50. 访问一个网站的流程
51. 三次握手，四次挥手
52. 什么是动态资源，什么是静态资源
53. worker支持的最大并发数是什么?
54. Tomcat和Resin有什么区别，工作中你怎么选择?
55. 什么叫网站灰度发布?

56.. 统计ip访问情况，要求分析nginx访问日志，找出访问页面数量在前十位的ip
57. nginx各个版本的区别
58. nginx最新版本
59. 关于nginx access模块的面试题
60. nginx默认配置文件
61. location的规则
62. 配置nginx防盗链
63. drop，delete和truncate删除数据的区别?
64. MySQL主从原理
65. MySQL主从复制存在哪些问题?
66. MySQL复制的方法
67. 主从延迟产生的原因及解决方案?
68. 判断主从延迟的方法
69. MySQL忘记root密码如何找回
70. MySQL的数据备份方式
71. innodb的特性
72. varchar(100) 和varchar(200）的区别
73. MySQL主要的索引类型
74. 请说出非关系型数据库的典型产品、特点及应用场景?
75. 如何加强MySQL安全，请给出可行的具体措施?
76. Binlog工作模式有哪些？各什么特点，企业如何选择?
77. 生产一主多从从库宕机，如何手工恢复?
78. MySQL中MyISAM与InnoDB的区别，至少5点
79. 网站打开慢，请给出排查方法，如是数据库慢导致，如何排查并解决，请分析并举例?
80. xtrabackup的备份，增量备份及恢复的工作原理
81.误执行drop数据，如何通过xtrabackup恢复?
82. 如何做主从数据一致性校验?
83. MySQL有多少日志
84. MySQL binlog的几种日志录入格式以及区别
85. MySQL数据库cpu飙升到500%的话他怎么处理?
86. redis是单线程还是多线程?
87. redis常用的版本是?
88. redis 的使用场景?
89. redis常见的数据结构
90. redis持久化你们怎么做的?
91. 主从复制实现的原理
92. redis哨兵模式原理
93. memcache和redis的区别
94. redis有哪些架构模式?
95. 缓存雪崩?
96. 缓存穿透
97. 缓存击穿
98. redis为什么这么快
99. memcache有哪些应用场景
100. memcache 服务特点及工作原理
101. memcached是如何做身份验证的?
102. mongoDB是什么?
103. mongodb的优势
104. mongodb使用场景
105. kafka 中的ISR，AR代表什么，ISR伸缩又代表什么
106.kafka中的broker 是干什么的
107. kafka中的 zookeeper 起到什么作用，可以不用zookeeper么
108. kafka follower如何与leader同步数据
109. kafka 为什么那么快
110. Kafka中的消息是否会丢失和重复消费?
111. 为什么Kafka不支持读写分离?
112. 什么是消费者组?
113. Kafka 中的术语

1. 磁盘使用率检测（用shell脚本）

```
root@ecs-c13b ~]# cat fdisk.sh
#!/bin/bash
# 截取IP
IP=`ifconfig eth0 |awk -F " " 'NR==2{print $2}'`
# 定义使用率,并转换为数字
SPACE=`df -Ph |awk '{print int($5)}'`

for i in $SPACE
do
if [ $i -ge 90 ]
then
    echo "$IP的磁盘使用率已经超过了90%，请及时处理"

fi
done
```

2. LVS 负载均衡有哪些策略?

LVS一共有三种工作模式： DR，Tunnel,NAT

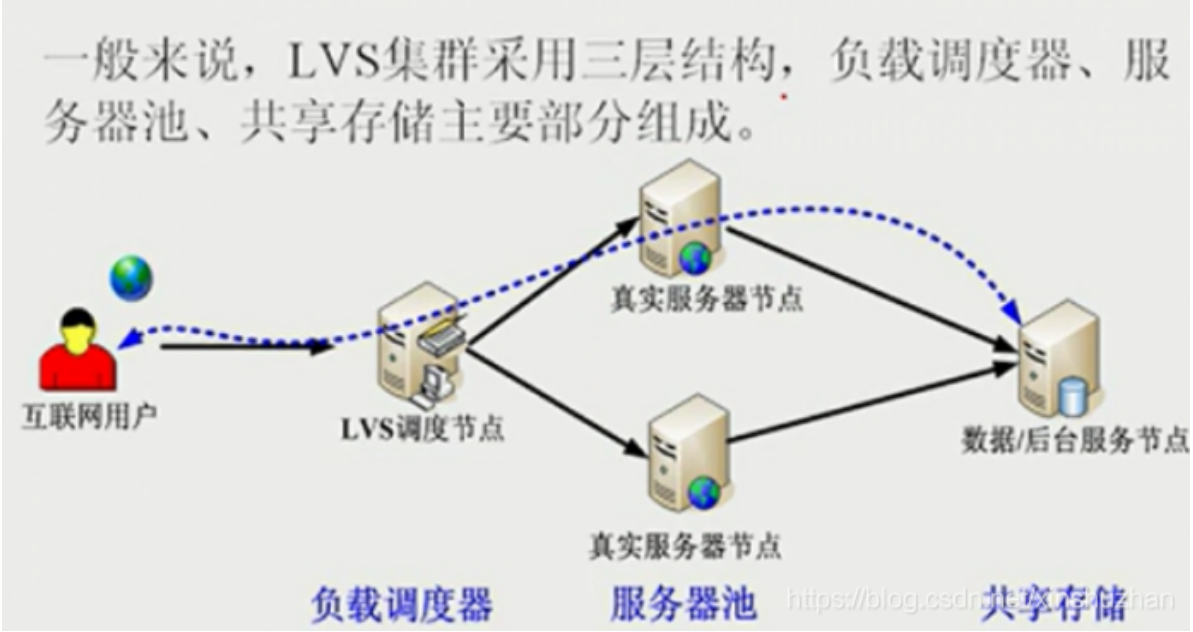
3. 谈谈你对LVS的理解?

LVS是一个虚拟的服务器集群系统，在unix系统下实现负载均衡的功能；采用IP负载均衡技术和机遇内容请求分发技术来实现。
LVS采用三层结构，分别是：
第一层： 负载调度器
第二层： 服务池
第三层：共享存储

负载调度器（load balancer/ Director），是整个集群的总代理，它有两个网卡，一个网卡面对访问网站的客户端，一个网卡面对整个集群的内部。负责将客户端的请求发送到一组服务器上执行，而客户也认为服务是来自这台主的。举个生动的例子，集群是个公司，负载调度器就是在外接揽生意，将接揽到的生意分发给后台的真正干活的真正的主机们。当然需要将活按照一定的算法分发下去，让大家都公平的干活。

服务器池（server pool/ Realserver），是一组真正执行客户请求的服务器，可以当做WEB服务器。就是上面例子中的小员工。

共享存储（shared storage），它为服务器池提供一个共享的存储区，这样很容易使得服务器池拥有相同的内容，提供相同的服务。一个公司得有一个后台账目吧，这才能协调。不然客户把钱付给了A，而换B接待客户，因为没有相同的账目。B说客户没付钱，那这样就不是客户体验度的问题了。



4. 负载均衡的原理是什么?

当客户端发起请求时，请求直接发给Director Server（调度器），这时会根据设定的调度算法，将请求按照算法的规定智能的分发到真正的后台服务器。以达到将压力均摊。

但是我们知道，http的连接时无状态的，假设这样一个场景，我登录某宝买东西，当我看上某款商品时，我将它加入购物车，但是我刷新了一下页面，这时由于负载均衡的原因，调度器又选了新的一台服务器为我提供服务，我刚才的购物车内容全都不见了，这样就会有十分差的用户体验。

所以就还需要一个存储共享，这样就保证了用户请求的数据是一样的

5. LVS由哪两部分组成的?

LVS 由2部分程序组成，包括 ipvs 和 ipvsadm。

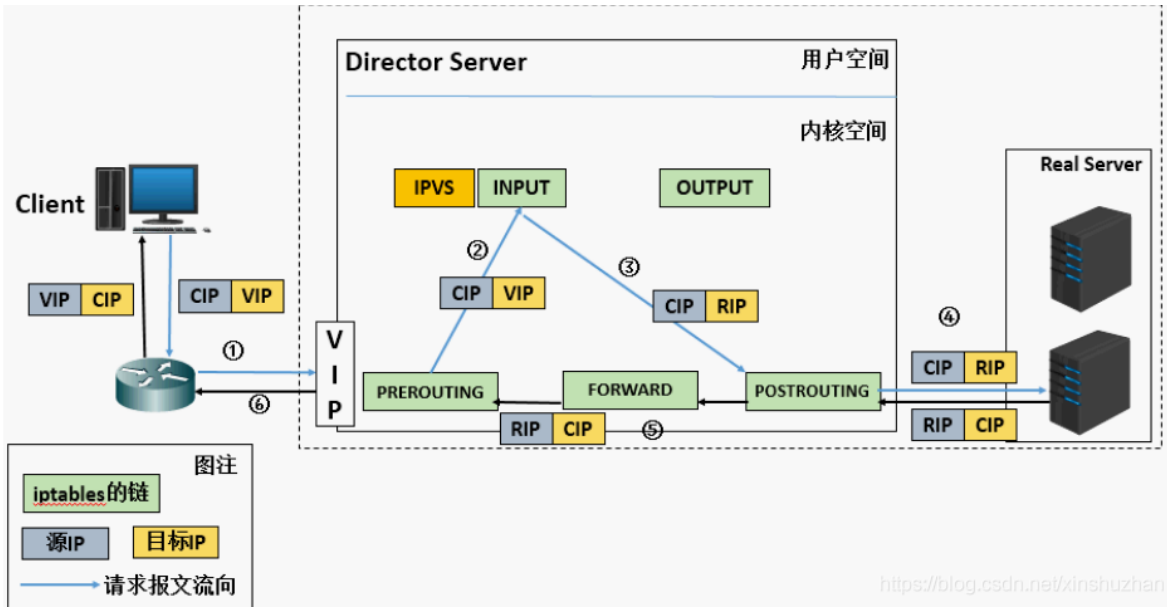
1.ipvs(ip virtual server)：一段代码工作在内核空间，叫ipvs，是真正生效实现调度的代码。

2. ipvsadm：另外一段是工作在用户空间，叫ipvsadm，负责为ipvs内核框架编写规则，定义谁是集群服务，而谁是后端真实的服务器(Real Server)

6. 与lvs相关的术语有哪些?

DS：Director Server。指的是前端负载均衡器节点。
RS：Real Server。后端真实的工作服务器。
VIP：Virtual IP 向外部直接面向用户请求，作为用户请求的目标的IP地址。
DIP：Director Server IP，主要用于和内部主机通讯的IP地址。
RIP：Real Server IP，后端服务器的IP地址。
CIP：Client IP，访问客户端的IP地址。

7. LVS-NAT模式的原理



> (a). 当用户请求到达Director Server，此时请求的数据报文会先到内核空间的PREROUTING链。此时报文的源IP为CIP，目标IP为VIP
> (b). PREROUTING检查发现数据包的目标IP是本机，将数据包送至INPUT链
> ©. IPVS比对数据包请求的服务是否为集群服务，若是，修改数据包的目标IP地址为后端服务器IP， 然后将数据包发至POSTROUTING链。 此时报文的源IP为CIP，目标IP为RIP
> (d). POSTROUTING链通过选路，将数据包发送给Real Server
> (e). Real Server比对发现目标为自己的IP，开始构建响应报文发回给Director Server。 此时报文的源IP为RIP，目标IP为CIP
> (f). Director Server在响应客户端前，此时会将源IP地址修改为自己的VIP地址，然后响应给客户端。 此时报文的源IP为VIP，目标IP为CIP

8. LVS-NAT模型的特性

RS应该使用私有地址，RS的网关必须指向DIP

DIP和RIP必须在同一个网段内

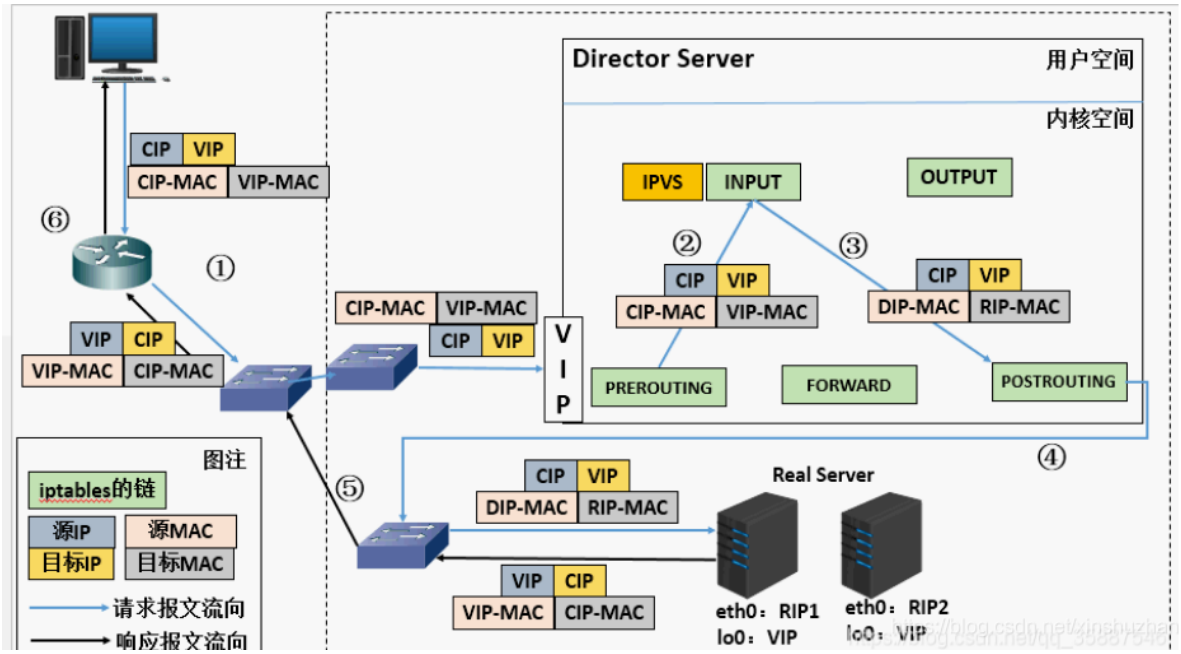
请求和响应报文都需要经过Director Server，高负载场景中，Director Server易成为性能瓶颈

支持端口映射

RS可以使用任意操作系统

缺陷：对Director Server压力会比较大，请求和响应都需经过director server

9. LVS-DR模式原理



> (a) 当用户请求到达Director Server，此时请求的数据报文会先到内核空间的PREROUTING链。 此时报文的源IP为CIP，目标IP为VIP
> (b) PREROUTING检查发现数据包的目标IP是本机，将数据包送至INPUT链
> © IPVS比对数据包请求的服务是否为集群服务，若是，将请求报文中的源MAC地址修改为DIP的MAC地址，将目标MAC地址修改RIP的MAC地址，然后将数据包发至POSTROUTING链。 此时的源IP和目的IP均未修改，仅修改了源MAC地址为DIP的MAC地址，目标MAC地址为RIP的MAC地址
> (d) 由于DS和RS在同一个网络中，所以是通过二层来传输。POSTROUTING链检查目标MAC地址为RIP的MAC地址，那么此时数据包将会发至Real Server。
> (e) RS发现请求报文的MAC地址是自己的MAC地址，就接收此报文。处理完成之后，将响应报文通过lo接口传送给eth0网卡然后向外发出。 此时的源IP地址为VIP，目标IP为CIP
> (f) 响应报文最终送达至客户端

10. LVS-DR模型的特性

特点1：保证前端路由将目标地址为VIP报文统统发给Director Server，而不是RS

RS可以使用私有地址；也可以是公网地址，如果使用公网地址，此时可以通过互联网对RIP进行直接访问

RS跟Director Server必须在同一个物理网络中

所有的请求报文经由Director Server，但响应报文必须不能进过Director Server

不支持地址转换，也不支持端口映射

RS可以是大多数常见的操作系统

RS的网关绝不允许指向DIP(因为我们不允许他经过director)

RS上的lo接口配置VIP的IP地址

缺陷：RS和DS必须在同一机房中

11. LVS三种负载均衡模式的比较

三种负载均衡： nat，tunneling，dr

类目	NAT	TUN	DR
操作系统	任意	支持隧道	多数（支持non-arp）
服务器网络	私有网络	局域网/广域网	局域网
服务器数目	10-20	100	大于100
服务器网关	负载均衡器	自己的路由	自己的路由
效率|一般|高|最高

12. LVS的负载调度算法

- 轮叫调度
- 加权轮叫调度
- 最小连接调度
- 加权最小连接调度
- 基于局部性能的最少连接
- 带复制的基于局部性能最小连接
- 目标地址散列调度
- 源地址散列调度

13. LVS与nginx的区别

lvs的优势（互联网老辛）：

- 1. 抗负载能力强，因为lvs工作方式的逻辑是非常简单的，而且工作在网络的第4层，仅作请求分发用，没有流量，所以在效率上基本不需要太过考虑。lvs一般很少出现故障，即使出现故障一般也是其他地方（如内存、CPU等）出现问题导致lvs出现问题。
- 2.配置性低，这通常是一大劣势同时也是一大优势，因为没有太多的可配置的选项，所以除了增减服务器，并不需要经常去触碰它，大大减少了人为出错的几率。
- 3.工作稳定，因为其本身抗负载能力很强，所以稳定性高也是顺理成章的事，另外各种lvs都有完整的双机热备方案，所以一点不用担心均衡器本身会出什么问题，节点出现故障的话，lvs会自动判别，所以系统整体是非常稳定的。
- 4.无流量，lvs仅仅分发请求，而流量并不从它本身出去，所以可以利用它这点来做一些线路分流之用。没有流量同时也保住了均衡器的IO性能不会受到大流量的影响。
- 5.lvs基本上能支持所有应用，因为lvs工作在第4层，所以它可以对几乎所有应用做负载均衡，包括http、数据库、聊天室等。

nginx与LVS的对比：

- nginx工作在网络的第7层，所以它可以针对http应用本身来做分流策略，比如针对域名、目录结构等，相比之下lvs并不具备这样的功能，所以nginx单凭这点可以利用的场合就远多于lvs了；但nginx有用的这些功能使其可调整度要高于lvs，所以经常要去触碰，由lvs的第2条优点来看，触碰多了，人为出现问题的几率也就会大。
- nginx对网络的依赖较小，理论上只要ping得通，网页访问正常，nginx就能连得通，nginx同时还能区分内外网，如果是同时拥有内外网的节点，就相当于单机拥有了备份线路；lvs就比较依赖于网络环境，目前来看服务器在同一网段内并且lvs使用direct方式分流，效果较能得到保证。另外注意，lvs需要向托管商至少申请多于一个ip来做visual ip。
- nginx安装和配置比较简单，测试起来也很方便，因为它基本能把错误用日志打印出来。lvs的安装和配置、测试就要花比较长的时间，因为同上所述，lvs对网络依赖性比较大，很多时候不能配置成功都是因为网络问题而不是配置问题，出了问题要解决也相应的会麻烦的多。

- nginx也同样能承受很高负载且稳定，但负载度和稳定度差lvs还有几个等级：nginx处理所有流量所以受限于机器IO和配置；本身的bug也还是难以避免的；nginx没有现成的双机热备方案，所以跑在单机上还是风险比较大，单机上的事情全都很难说。
- nginx可以检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点。目前lvs中ldirectd也能支持针对服务器内部的情况来监控，但lvs的原理使其不能重发请求。比如用户正在上传一个文件，而处理该上传的节点刚好在上传过程中出现故障，nginx会把上传切到另一台服务器重新处理，而lvs就直接断掉了。

两者配合使用：

nginx用来做http的反向代理，能够upsteam实现http请求的多种方式的均衡转发。由于采用的是异步转发可以做到如果一个服务器请求失败，立即切换到其他服务器，直到请求成功或者最后一台服务器失败为止。这可以最大程度的提高系统的请求成功率。

lvs采用的是同步请求转发的策略。这里说一下同步转发和异步转发的区别。同步转发是在lvs服务器接收到请求之后，立即redirect到一个后端服务器，由客户端直接和后端服务器建立连接。异步转发是nginx在保持客户端连接的同时，发起一个相同内容的新请求到后端，等后端返回结果后，由nginx返回给客户端。

进一步来说：当做为负载均衡服务器的nginx和lvs处理相同的请求时，所有的请求和响应流量都会经过nginx；但是使用lvs时，仅请求流量经过lvs的网络，响应流量由后端服务器的网络返回。

也就是，当作为后端的服务器规模庞大时，nginx的网络带宽就成了一个巨大的瓶颈。

但是仅仅使用lvs作为负载均衡的话，一旦后端接受到请求的服务器出了问题，那么这次请求就失败了。但是如果在lvs的后端在添加一层nginx（多个），每个nginx后端再有几台应用服务器，那么结合两者的优势，既能避免单nginx的流量集中瓶颈，又能避免单lvs时一锤子买卖的问题。

14. 负载均衡的作用有哪些?

1、转发功能

按照一定的算法【权重、轮询】，将客户端请求转发到不同应用服务器上，减轻单个服务器压力，提高系统并发量。

2、故障移除

通过心跳检测的方式，判断应用服务器当前是否可以正常工作，如果服务器期宕掉，自动将请求发送到其他应用服务器。

3、恢复添加

如检测到发生故障的应用服务器恢复工作，自动将其添加到处理用户请求队伍中。

15. nginx实现负载均衡的分发策略

Nginx 的 upstream目前支持的分配算法：
1)、轮询 ——1：1 轮流处理请求（默认）

> 每个请求按时间顺序逐一分配到不同的应用服务器，如果应用服务器down掉，自动剔除，剩下的继续轮询。

2)、权重 ——you can you up
通过配置权重，指定轮询几率，权重和访问比率成正比，用于应用服务器性能不均的情况。
3)、ip_哈希算法
每个请求按访问ip的hash结果分配，这样每个访客固定访问一个应用服务器，可以解决session共享的问题。

16. keepalived 是什么?

广义上讲是高可用，狭义上讲是主机的冗余和管理

Keepalived起初是为LVS设计的，专门用来监控集群系统中各个服务节点的状态，它根据TCP/IP参考模型的第三、第四层、第五层交换机制检测每个服务节点的状态，如果某个服务器节点出现异常，或者工作出现故障，Keepalived将检测到，并将出现的故障的服务器节点从集群系统中剔除，这些工作全部是自动完成的，不需要人工干涉，需要人工完成的只是修复出现故障的服务节点。

后来Keepalived又加入了VRRP的功能，VRRP（VritrualRouterRedundancyProtocol,虚拟路由冗余协议)出现的目的是解决静态路由出现的单点故障问题，通过VRRP可以实现网络不间断稳定运行，因此Keepalvied一方面具有服务器状态检测和故障隔离功能，另外一方面也有HAcluster功能。

所以keepalived的核心功能就是健康检查和失败且换。
所谓的健康检查，就是采用tcp三次握手，icmp请求，http请求，udp echo请求等方式对负载均衡器后面的实际的服务器(通常是承载真实业务的服务器)进行保活；

而失败切换主要是应用于配置了主备模式的负载均衡器，利用VRRP维持主备负载均衡器的心跳，当主负载均衡器出现问题时，由备负载均衡器承载对应的业务，从而在最大限度上减少流量损失，并提供服务的稳定性

17. 你是如何理解VRRP协议的

为什么使用VRRP?
主机之间的通信都是通过配置静态路由或者(默认网关)来完成的，而主机之间的路由器一旦发生故障，通信就会失效，因此这种通信模式当中，路由器就成了一个单点瓶颈，为了解决这个问题，就引入了VRRP协议。

VRRP协议是一种容错的主备模式的协议，保证当主机的下一跳路由出现故障时，由另一台路由器来代替出现故障的路由器进行工作，通过VRRP可以在网络发生故障时透明的进行设备切换而不影响主机之间的数据通信。

VRRP的三种状态：
VRRP路由器在运行过程中有三种状态：

1. Initialize状态： 系统启动后就进入Initialize，此状态下路由器不对VRRP报文做任何处理；
2. Master状态；
3. Backup状态；
   一般主路由器处于Master状态，备份路由器处于Backup状态。

18. keepalived的工作原理?

keepalived采用是模块化设计，不同模块实现不同的功能。

keepalived主要有三个模块，分别是core、check和vrrp。

core：是keepalived的核心，负责主进程的启动和维护，全局配置文件的加载解析等

check： 负责healthchecker(健康检查)，包括了各种健康检查方式，以及对应的配置的解析包括LVS的配置解析；可基于脚本检查对IPVS后端服务器健康状况进行检查

vrrp：VRRPD子进程，VRRPD子进程就是来实现VRRP协议的

Keepalived高可用对之间是通过 VRRP进行通信的， VRRP是通过竞选机制来确定主备的，主的优先级高于备，因此，工作时主会优先获得所有的资源，备节点处于等待状态，当主宕机的时候，备节点就会接管主节点的资源，然后顶替主节点对外提供服务

在Keepalived服务对之间，只有作为主的服务器会一直发送 VRRP广播包,告诉备它还活着，此时备不会抢占主，当主不可用时，即备监听不到主发送的广播包时，就会启动相关服务接管资源，保证业务的连续性.接管速度最快

19. 出现脑裂的原因

什么是脑裂?

- 在高可用（HA）系统中，当联系2个节点的“心跳线”断开时，本来为一整体、动作协调的HA系统，就分裂成为2个独立的个体。
- 由于相互失去了联系，都以为是对方出了故障。两个节点上的HA软件像“裂脑人”一样，争抢“共享资源”、争起“应用服务”，就会发生严重后果。共享资源被瓜分、两边“服务”都起不来了；或者两边“服务”都起来了，但同时读写“共享存储”，导致数据损坏

都有哪些原因导致脑裂?

高可用服务器对之间心跳线链路发生故障，导致无法正常通信。

因心跳线坏了（包括断了，老化）。

因网卡及相关驱动坏了，ip配置及冲突问题（网卡直连）

因心跳线间连接的设备故障（网卡及交换机）

因仲裁的机器出问题（采用仲裁的方案）

高可用服务器上开启了 iptables防火墙阻挡了心跳消息传输。

高可用服务器上心跳网卡地址等信息配置不正确，导致发送心跳失败

其他服务配置不当等原因，如心跳方式不同，心跳广插冲突、软件Bug等。

20. 如何解决keepalived脑裂问题?

在实际生产环境中，我们从以下方面防止脑裂：

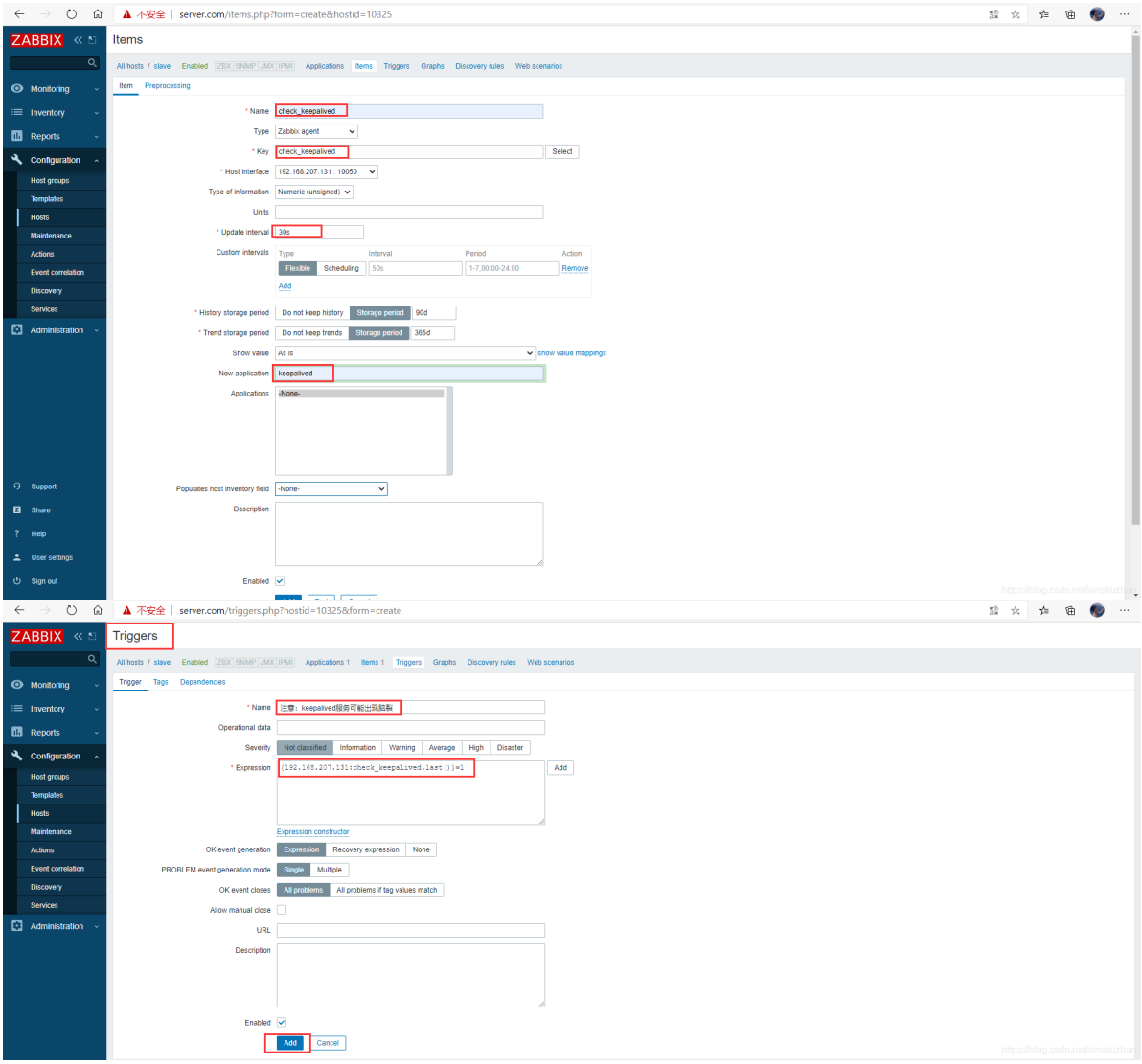
- 同时使用串行电缆和以太网电缆连接、同时使用两条心跳线路，这样一条线路断了，另外一条还是好的，依然能传送心跳消息
- 当检查脑裂时强行关闭一个心跳节点（这个功能需要特殊设备支持，如stonith、fence）相当于备节点接收不到心跳消息，通过单独的线路发送关机命令关闭主节点的电源
- 做好对脑裂的监控报警
  解决常见方案：
- 如果开启防火墙，一定要让心跳消息通过，一般通过允许IP段的形式解决
- 可以拉一条以太网网线或者串口线作为主被节点心跳线路的冗余
- 开发检测程序通过监控软件检测脑裂

21. zabbix如何监控脑裂?

监控只是监控发生脑裂的可能性，不能保证一定是发生了脑裂，因为正常的主备切换VIP也是会到备上的

监控脚本：

```
[root@slave ~]# mkdir -p /scripts && cd /scripts
[root@slave scripts]# vim check_keepalived.sh
#!/bin/bash

if [ `ip a show ens33 |grep 192.168.32.250|wc -l` -ne 0 ]
then
    echo "keepalived is error!"
else
    echo "keepalived is OK !"
fi
```



22. nginx做负载均衡实现的策略有哪些

- 轮询（默认）
- 权重
- ip_hash
- fair(第三方插件)
- url_hash（第三方插件）

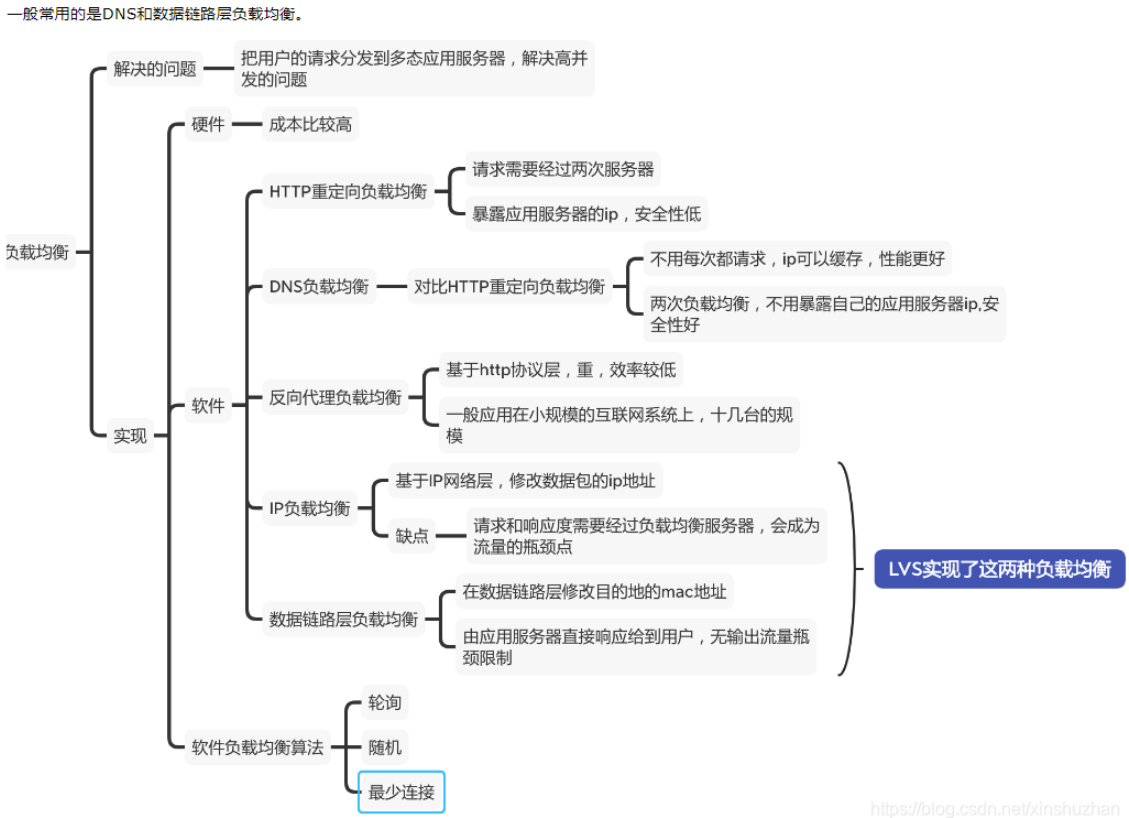
23. nginx做负载均衡用到哪些模块

upstream 定义负载节点池。
location 模块 进行URL匹配。
proxy模块 发送请求给upstream定义的节点池。

24. 负载均衡有哪些实现方式

- 硬件负载
- HTTP重定向负载均衡
- DNS负载均衡
- 反向代理负载均衡
- IP层负载均衡
- 数据链路层负载均衡

一般常用的是DNS和数据链路层负载均衡。

- 

25. nginx如何实现四层负载?

四层负载分为动态和静态负载
Nginx的四层静态负载均衡需要启用ngx_stream_core_module模块
默认情况下，ngx_stream_core_module是没有启用的，需要在安装Nginx时，添加--with-stream配置参数启用
配置HTTP负载均衡时，都是配置在http指令下，配置四层负载均衡，则是在stream指令下，结构如下所示.

```
stream {
 upstream mysql_backend {
  server 192.168.175.100:3306 max_fails=2 fail_timeout=10s weight=1;
  least_conn;
 }
 server { #监听端口，默认使用的是tcp协议，如果需要UDP协议，则配置成listen 3307 udp;
listen 3307; #失败重试 proxy_next_upstream on;
  proxy_next_upstream_timeout 0;
  proxy_next_upstream_tries 0; #超时配置 #配置与上游服务器连接超时时间，默认60s
proxy_connect_timeout 1s; #配置与客户端上游服务器连接的两次成功读/写操作的超时时间，如果超
时，将自动断开连接 #即连接存活时间，通过它可以释放不活跃的连接，默认10分钟 proxy_timeout 1m;
#限速配置 #从客户端读数据的速率，单位为每秒字节数，默认为0，不限速 proxy_upload_rate 0; #从
上游服务器读数据的速率，单位为每秒字节数，默认为0，不限速 proxy_download_rate 0; #上游服务器
proxy_pass mysql_backend;
 }
}
```

使用Nginx的四层动态负载均衡有两种方案：使用商业版的Nginx和使用开源的nginx-stream-upsync-module模块。注意：四层动态负载均衡可以使用nginx-stream-upsync-module模块，七层动态负载均衡可以使用nginx-upsync-module模块。

26. 你知道的web服务有哪些?

- apache
- nginx
- IIS
- tomcat
- lighttpd
- weblogic

27. 为什么要用nginx

- 跨平台、配置简单，非阻塞、高并发连接：处理2-3万并发连接数，官方监测能支持5万并发，
- 内存消耗小：开启10个nginx才占150M内存 ，nginx处理静态文件好,耗费内存少，
- 内置的健康检查功能：如果有一个服务器宕机，会做一个健康检查，再发送的请求就不会发送到宕机的服务器了。重新将请求提交到其他的节点上。
- 节省宽带：支持GZIP压缩，可以添加浏览器本地缓存
- 稳定性高：宕机的概率非常小
- 接收用户请求是异步的

28 . nginx的性能为什么比apache高?

nginx采用的是epoll模型和kqueue网络模型，而apache采用的是select模型

举一个例子来解释两种模型的区别：
菜鸟驿站放着很多快件，以前去拿快件都是短信通知你有快件，然后你去了之后，负责菜鸟驿站的人在一堆快递里帮你找，直到找到为止。

但现在菜鸟驿站的方式变了，他会发你一个地址，比如 3-3-5009. 这个就是第三个货架的第三排，从做往右第九个。

如果有几百个人同时去找快递，这两种方式哪个更有效率，不言而喻。

之前还看到这个例子也比较形象：

> 假设你在大学读书，住的宿舍楼有很多间房间，你的朋友要来找你。

select版宿管大妈就会带着你的朋友挨个房间去找，直到找到你为止。

而epoll版宿管大妈会先记下每位同学的房间号，

你的朋友来时，只需告诉你的朋友你住在哪个房间即可，不用亲自带着你的朋友满大楼找人。

如果来了10000个人，都要找自己住这栋楼的同学时，select版和epoll版宿管大妈，谁的效率更高，不言自明。

同理，在高并发服务器中，轮询I/O是最耗时间的操作之一，select和epoll的性能谁的性能更高，同样十分明了

select 采用的是轮询的方式来处理请求，轮询的次数越多，耗时也就越多。

29 . epoll的组成

epoll的接口非常简单，一共就三个函数：

1. int epoll_create(int size);
创建一个epoll的句柄，size用来告诉内核这个监听的数目一共有多大。

这个参数不同于select()中的第一个参数，给出最大监听的fd+1的值。

需要注意的是，当创建好epoll句柄后，它就是会占用一个fd值，在linux下如果查看/proc/进程id/fd/，

是能够看到这个fd的，所以在使用完epoll后，必须调用close()关闭，否则可能导致fd被耗尽。

2. int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
epoll的事件注册函数，它不同与select()是在监听事件时告诉内核要监听什么类型的事件，

而是在这里先注册要监听的事件类型。第一个参数是epoll_create()的返回值，

第二个参数表示动作，用三个宏来表示：

EPOLL_CTL_ADD：注册新的fd到epfd中；

EPOLL_CTL_MOD：修改已经注册的fd的监听事件；

EPOLL_CTL_DEL：从epfd中删除一个fd；

第三个参数是需要监听的fd，第四个参数是告诉内核需要监听什么事

3. int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout);
等待事件的产生，类似于select()调用。

参数events用来从内核得到事件的集合，maxevents告之内核这个events有多大，这个 maxevents的值不能大于创建epoll_create()时的size，参数timeout是超时时间（毫秒，0会立即返回，-1将不确定，也有说法说是永久阻塞）。

该函数返回需要处理的事件数目，如返回0表示已超时

30 . nginx和apache的区别

Nginx

- 轻量级，采用 C 进行编写，同样的 web 服务，会占用更少的内存及资源
- 抗并发，nginx 以 epoll and kqueue 作为开发模型，处理请求是异步非阻塞的，负载能力比 apache 高很多，而 apache 则是阻塞型的。在高并发下 nginx 能保持低资源低消耗高性能 ，而 apache 在 PHP 处理慢或者前端压力很大的情况下，很容易出现进程数飙升，从而拒绝服务的现象。
- nginx 处理静态文件好，静态处理性能比 apache 高三倍以上
- nginx 的设计高度模块化，编写模块相对简单
- nginx 配置简洁，正则配置让很多事情变得简单，而且改完配置能使用 -t 测试配置有没有问题，apache 配置复杂 ，重启的时候发现配置出错了，会很崩溃
- nginx 作为负载均衡服务器，支持 7 层负载均衡
  七层负载可以有效的防止ddos攻击
- nginx本身就是一个反向代理服务器，也可以左右邮件代理服务器来使用

Apache

- apache 的 rewrite 比 nginx 强大，在 rewrite 频繁的情况下，用 apache
- apache 发展到现在，模块超多，基本想到的都可以找到
- apache 更为成熟，少 bug ， nginx 的 bug 相对较多
- apache 对 PHP 支持比较简单，nginx 需要配合其他后端用
- apache 在处理动态请求有优势，nginx 在这方面是鸡肋，一般动态请求要 apache 去做，nginx 适合静态和反向。
- apache 仍然是目前的主流，拥有丰富的特性，成熟的技术和开发社区

两者最核心的区别在于 apache 是同步多进程模型，一个连接对应一个进程，而 nginx 是异步的，多个连接（万级别）可以对应一个进程。

需要稳定用apache，需要高性能用nginx

31. Tomcat作为web的优缺点?

缺点：
tomcat 只能用做java服务器，处理静态请求的能力不如nginx和apache。，高并发能力有限
优点：动态解析容器，处理动态请求，是编译JSP/Servlet的容器，轻量级

32. tomcat的三个端口及作用

8005： 关闭Tomcat通信接口
8009： 与其他httpd服务器通信接口，用于http服务器的集合
8080： 建立httpd连接用，如浏览器访问

33. fastcgi 和cgi的区别

cgi:
web 服务器会根据请求的内容，然后会 fork 一个新进程来运行外部 c 程序（或 perl 脚本...）， 这个进程会把处理完的数据返回给 web 服务器，最后 web 服务器把内容发送给用户，刚才 fork 的进程也随之退出。

如果下次用户还请求改动态脚本，那么 web 服务器又再次 fork 一个新进程，周而复始的进行。

fastcgi
web 服务器收到一个请求时，他不会重新 fork 一个进程（因为这个进程在 web 服务器启动时就开启了，而且不会退出），web 服务器直接把内容传递给这个进程（进程间通信，但 fastcgi 使用了别的方式，tcp 方式通信），这个进程收到请求后进行处理，把结果返回给 web 服务器，最后自己接着等待下一个请求的到来，而不是退出。

34. nginx常用的命令

```
启动 nginx 。
停止 nginx -s stop 或 nginx -s quit 。
重载配置 ./sbin/nginx -s reload(平滑重启) 或 service nginx reload 。
重载指定配置文件 .nginx -c /usr/local/nginx/conf/nginx.conf 。
查看 nginx 版本 nginx -v 。
检查配置文件是否正确 nginx -t 。
显示帮助信息 nginx -h 。
```

35. 什么是反向代理，什么是正向代理，以及区别?

正向代理：
所谓的正向代理就是： 需要在用户端去配置的。配置完再去访问具体的服务，这叫正向代理

正向代理，其实是"代理服务器"代理了"客户端"，去和"目标服务器"进行交互。
正向代理的用途：

- 提高访问速度
- 隐藏客户真实IP

反向代理：

反向代理是 在服务端的，不需要访问用户关心。用户访问服务器A, A服务器是代理服务器，将用户服务再转发到服务器B.这就是反向代理

反向代理的作用：
1.缓存，将服务器的响应缓存在自己的内存中，减少服务器的压力。
2.负载均衡，将用户请求分配给多个服务器。
3.访问控制

36. Squid、Varinsh、Nginx 有什么区别?

三者都实现缓存服务器的作用

- Nginx本来是反向代理/web服务器，用了插件可以做做这个副业(缓存服务器)。但本身支持的特性不是很多，只能缓存静态文件
- varinsh 和squid是专业的cache服务，而nginx这些需要使用第三方模块
- varnish本身在技术上的优势要高于squid，它采用了可视化页面缓存技术。

```
在内存的利用上，Varnis h比 Squid 具有优势，性能要比 Squid 高。
  还有强大的通过 Varnish 管理端口，可以使用正则表达式快速、批量地清除部分缓存
   Varnish 是内存缓存，速度一流，但是内存缓存也限制了其容量，缓存页面和图片一般是挺好的。
```

要做 cache 服务的话，我们肯定是要选择专业的 cache 服务，优先选择Squid 或者 Varnish

37. nginx是如何处理http请求的

四个步骤：
读取解析请求行；
读取解析请求头；
开始最重要的部分，即多阶段处理；

> nginx把请求处理划分成了11个阶段，也就是说当nginx读取了请求行和请求头之后，将请求封装了结构体ngx_http_request_t，然后每个阶段的handler都会根据这个ngx_http_request_t，对请求进行处理，例如重写uri，权限控制，路径查找，生成内容以及记录日志等等；

最后将结果放回给客户单。

也可以这么回答：

- 首先，Nginx 在启动时，会解析配置文件，得到需要监听的端口与 IP 地址，然后在 Nginx 的 Master 进程里面先初始化好这个监控的Socket(创建 S ocket，设置 addr、reuse 等选项，绑定到指定的 ip 地址端口，再 listen 监听)。
- 然后，再 fork(一个现有进程可以调用 fork 函数创建一个新进程。由 fork 创建的新进程被称为子进程)出多个子进程出来。
- 之后，子进程会竞争 accept 新的连接。此时，客户端就可以向 nginx 发起连接了。当客户端与 nginx进行三次握手，与 nginx 建立好一个连接后。此时，某一个子进程会 accept 成功，得到这个建立好的连接的 Socket ，然后创建 nginx 对连接的封装，即 ngx_connection_t 结构体。

-接着，设置读写事件处理函数，并添加读写事件来与客户端进行数据的交换。

- 最后，Nginx 或客户端来主动关掉连接，到此，一个连接就寿终正寝了。

38. nginx虚拟主机有哪些?

基于域名的虚拟主机
基于端口的虚拟主机
基于IP的虚拟主机

39. nginx怎么实现后端服务的健康检查

方式一，利用 nginx 自带模块 ngx_http_proxy_module 和 ngx_http_upstream_module 对后端节点做健康检查。

方式二，利用 nginx_upstream_check_module 模块对后端节点做健康检查。（推荐此方法）

40. apache中的Worker 和 Prefork 之间的区别是什么?

它们都是MPM, Worker 和 prefork 有它们各自在Apache上的运行机制. 它们完全依赖于你想要以哪一种模式启动你的Apache.

1.Worker 和 MPM基本的区别在于它们产生子进程的处理过程. 在Prefork MPM中, 一个主httpd进行被启动，这个主进程会管理所有其它子进程为客户端请求提供服务. 而在worker MPM中一个httpd进程被激活，则会使用不同的线程来为客户端请求提供服务.

2.Prefork MPM 使用多个子进程，每一个进程带有一个线程而 worker MPM 使用多个子进程，每一个进程带有多个线程.

3.Prefork MPM中的连接处理, 每一个进程一次处理一个连接而在Worker mpm中每一个线程一次处理一个连接.

4.内存占用 Prefork MPM 占用庞大的内存, 而Worker占用更小的内存.

41. Tomcat缺省端口是多少，怎么修改

- 找到Tomcat目录下的conf文件夹
- 进入conf文件夹里面找到server.xml文件
- 打开server.xml文件
- 在server.xml文件里面找到下列信息
- 把Connector标签的8080端口改成你想要的端口

42. Tomcat的工作模式是什么?

Tomcat作为servlet容器，有三种工作模式：

1、独立的servlet容器，servlet容器是web服务器的一部分；

2、进程内的servlet容器，servlet容器是作为web服务器的插件和java容器的实现，web服务器插件在内部地址空间打开一个jvm使得java容器在内部得以运行。反应速度快但伸缩性不足；

3、进程外的servlet容器，servlet容器运行于web服务器之外的地址空间，并作为web服务器的插件和java容器实现的结合。反应时间不如进程内但伸缩性和稳定性比进程内优；

进入Tomcat的请求可以根据Tomcat的工作模式分为如下两类：

- Tomcat作为应用程序服务器：请求来自于前端的web服务器，这可能是Apache, IIS, Nginx等；
- Tomcat作为独立服务器：请求来自于web浏览器；

43. Web请求在Tomcat请求中的请求流程是怎么样的?

- 浏览器输入URL地址;
- 查询本机hosts文件寻找IP;
- 查询DNS服务器寻找IP;
- 向该IP发送Http请求;
- Tomcat容器解析主机名;
- Tomcat容器解析Web应用;
- Tomcat容器解析资源名称;
- Tomcat容器获取资源;
- Tomcat响应浏览器。

44. 怎么监控Tomcat的内存使用情况

使用JDK自带的jconsole可以比较明了的看到内存的使用情况，线程的状态，当前加载的类的总量等；

JDK自带的jvisualvm可以下载插件（如GC等），可以查看更丰富的信息。如果是分析本地的Tomcat的话，还可以进行内存抽样等，检查每个类的使用情况
在java/bin目录下

45. nginx的优化你都做过哪些?

```
.    gzip压缩优化
.    expires缓存有还
.    网络IO事件模型优化
.    隐藏软件名称和版本号
.    防盗链优化
.    禁止恶意域名解析
.    禁止通过IP地址访问网站
.    HTTP请求方法优化
.    防DOS攻击单IP并发连接的控制，与连接速率控制
.   严格设置web站点目录的权限
```

. 将nginx进程以及站点运行于监牢模式
. 通过robot协议以及HTTP_USER_AGENT防爬虫优化
. 配置错误页面根据错误码指定网页反馈给用户
. nginx日志相关优化访问日志切割轮询，不记录指定元素日志、最小化日志目录权限
. 限制上传到资源目录的程序被访问，防止木马入侵系统破坏文件
. FastCGI参数buffer和cache配置文件的优化
. php.ini和php-fpm.conf配置文件的优化
. 有关web服务的Linux内核方面深度优化（网络连接、IO、内存等）
. nginx加密传输优化（SSL）
. web服务器磁盘挂载及网络文件系统的优化
. 使用nginx cache

一：配置文件中对优化有明显效果的：

1. worker_processes 8;
nginx 进程数，建议按照cpu 数目来指定，一般为它的倍数 (如,2个四核的cpu计为8)。

2. worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

为每个进程分配cpu，上例中将8 个进程分配到8 个cpu，当然可以写多个，或者将一个进程分配到多个cpu。

3. worker_rlimit_nofile 65535;

这个指令是指当一个nginx 进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx 进程数相除，但是nginx 分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。

现在在linux 2.6内核下开启文件打开数为65535，worker_rlimit_nofile就相应应该填写65535。

这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。

4. use epoll

5. worker_connections 65535
每个进程允许的最多连接数， 理论上每台nginx 服务器的最大连接数为worker_processes*worker_connections。

6. keepalive_timeout 60;

7. client_header_buffer_size 4k;
客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k

8. open_file_cache max=65535 inactive=60s;

这个将为打开文件指定缓存，默认是没有启用的，max 指定缓存数量，建议和打开文件数一致，inactive 是指经过多长时间文件没被请求后删除缓存。

9. open_file_cache_valid 80s;

这个是指多长时间检查一次缓存的有效信息。

10. open_file_cache_min_uses 1;

open_file_cache 指令中的inactive 参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive 时间内一次没被使用，它将被移除。

46. Tomcat你做过哪些优化

1. Tomcat的运行模式： bio,nio, apr
   一般使用nio模式，bio效率低，apr对系统配置有一些更高的要求
2. 关键配置
   maxThreads： 最大线程数，默认是200，
   minspareThread： 最小活跃线程数，默认是25
   maxqueuesize: 最大等待队列个数
3. 影响性能的配置：
   compression 设置成on，开启压缩
   禁用AJP连接器： 用nginx+Tomcat的架构，用不到AJP
   enableLookups=false 关闭反查域名，直接返回ip，提高效率
   disableUploadTimeou=false上传是否使用超时机制
   acceptCount=300 , 当前所有可以使用的处理请求都被使用时，传入请求连接最大队列长队，超过个数不予处理，默认是100
   keepalive timeout=120000 场链接保持时间
4. 优化jvm
   /bin/catalina.sh

> -server：jvm的server工作模式，对应的有client工作模式。使用"java -version"可以查看当前工作模式
>
> -Xms1024m：初始Heap大小，使用的最小内存
>
> -Xmx1024m：Java heap最大值，使用的最大内存。经验: 设置Xms大小等于Xmx大小
>
> -XX:NewSize=512m：表示新生代初始内存的大小，应该小于 -Xms的值
>
> -XX:MaxNewSize=1024M：表示新生代可被分配的内存的最大上限，应该小于 -Xmx的值
>
> -XX:PermSize=1024m：设定内存的永久保存区域,内存的永久保存区域，VM 存放Class 和 Meta 信息，JVM在运行期间不会清除该区域
> -XX:MaxPermSize=1024m：设定最大内存的永久保存区域。经验: 设置PermSize大小等于MaxPermSize大小
>
> -XX:+DisableExplicitGC：自动将System.gc() 调用转换成一个空操作，即应用中调用System.gc() 会变成一个空操作，避免程序员在代码里进行System.gc()这种危险操作。System.gc()
> 除非是到了万不得也的情况下使用，都应该交给 JVM。

47. nginx的session不同步怎么办

我们可以采用ip_hash指令解决这个问题，如果客户已经访问了某个服务器，当用户再次访问时，会将该请求通过哈希算法，自动定位到该服务器。即每个访客固定访问一个后端服务器，可以解决session的问题。

其他办法：那就是用spring_session+redis，把session放到缓存中实现session共享。

48. nginx的常用模块有哪些?

```
1、ngx_http_core_module #包括一些核心的http参数配置，对应Nginx的配置为HTTP区块部分

2、ngx_http_access_module #访问控制模块，用来控制网站用户对Nginx的访问

3、ngx_http_gzip_module #压缩模块，对Nginx返回的数据压缩，属于性能优化模块

4、ngx_http_fastcgi_module #FastCGI模块，和 动态应用相关的模块，例如PHP
```

```
5、ngx_http_proxy_module #Proxy代理模块

6、ngx_http_upstream_module #负载均衡模块，可以实现网站的负载均衡功能及节点的健康检查

7、ngx_http_rewrite_module #URL地址重写模块

8、ngx_http_limit_conn_module #限制用户并发连接数及请求数模块（防止ddos）

9、ngx_http_limit_req_module #根据定义的key限制Nginx请求过程的速率

10、ngx_http_log_module #访问日志模块，以指定的格式记录Nginx客户访问日志等信息

11、ngx_http_auth_basic_module #Web认证模块，设置Web用户通过账号、密码访问Nginx

12、ngx_http_ssl_module #ssl模块，用于加密的http连接，如https

13、ngx_http_stub_status_module #记录Nginx基本访问状态信息等模块
```

49. nginx常用状态码

```
200 （成功） 服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。
201 （已创建） 请求成功并且服务器创建了新的资源。
202 （已接受） 服务器已接受请求，但尚未处理。
203 （非授权信息） 服务器已成功处理了请求，但返回的信息可能来自另一来源。
204 （无内容） 服务器成功处理了请求，但没有返回任何内容。
205 （重置内容） 服务器成功处理了请求，但没有返回任何内容。
206 （部分内容） 服务器成功处理了部分 GET 请求。


300 （多种选择） 针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。
301 （永久移动） 请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。
302 （临时移动） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。
303 （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。
304 （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。
305 （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。
307 （临时重定向） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。


400 （错误请求） 服务器不理解请求的语法。
401 （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。
403 （禁止） 服务器拒绝请求。
404 （未找到） 服务器找不到请求的网页。
405 （方法禁用） 禁用请求中指定的方法。
406 （不接受） 无法使用请求的内容特性响应请求的网页。
407 （需要代理授权） 此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。
408 （请求超时） 服务器等候请求时发生超时。
409 （冲突） 服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。
410 （已删除） 如果请求的资源已永久删除，服务器就会返回此响应。
411 （需要有效长度） 服务器不接受不含有效内容长度标头字段的请求。
412 （未满足前提条件） 服务器未满足请求者在请求中设置的其中一个前提条件。
413 （请求实体过大） 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。
```

```
414 （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。
415 （不支持的媒体类型） 请求的格式不受请求页面的支持。
416 （请求范围不符合要求） 如果页面无法提供请求的范围，则服务器会返回此状态代码。
417 （未满足期望值） 服务器未满足"期望"请求标头字段的要求。


500 （服务器内部错误） 服务器遇到错误，无法完成请求。
501 （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。
502 （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。
503 （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。
504 （网关超时） 服务器作为网关或代理，但是没有及时从上游服务器收到请求。
505 （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。
```

50. 访问一个网站的流程

用户输入网站按回车， 查找本地缓存，如果有就打开页面，如果没有，利用DNS做域名解析，递归查询，一级一级的向上提交查询请求，知道查询到为止
HOSTS表--> 本地DNS -->上层DNS（包括根DNS）

经过了DNS解析，知道了网站的IP地址，然后建立tcp三次握手； 建立请求后，发送请求报文，默认请求的是index.html

传送完毕，断开连接

51. 三次握手，四次挥手

三次握手

01）由客户端（用户）发送建立TCP连接的请求报文，其中报文中包含seq序列号，是由发送端随机生成的。

并且还将报文中SYN字段置为1，表示需要建立TCP连接请求。

02）服务端（就是百度服务器）会回复客户端（用户）发送的TCP连接请求报文，其中包含seq序列号，也是由回复端随机生成的，

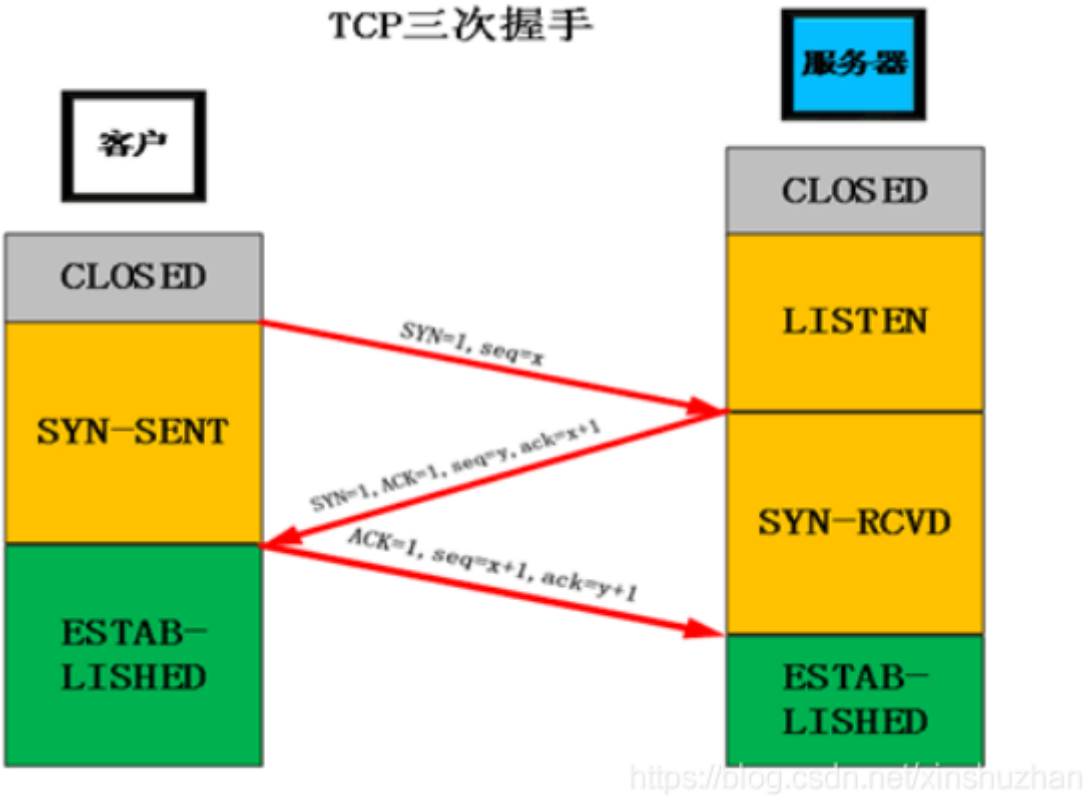
并且将回复报文的SYN字段置1，而且会产生ACK验证字段，ACK验证字段数值是在客户端发过来的seq序列号基础上加1进行回复：

并且还会回复ack确认控制字段，以便客户端收到信息时，知晓自己的TCP建立请求已得到了确认。

03）客户端收到服务端发送的TCP建立请求后，会使自己的原有序列号加1进行再次发送序列号，

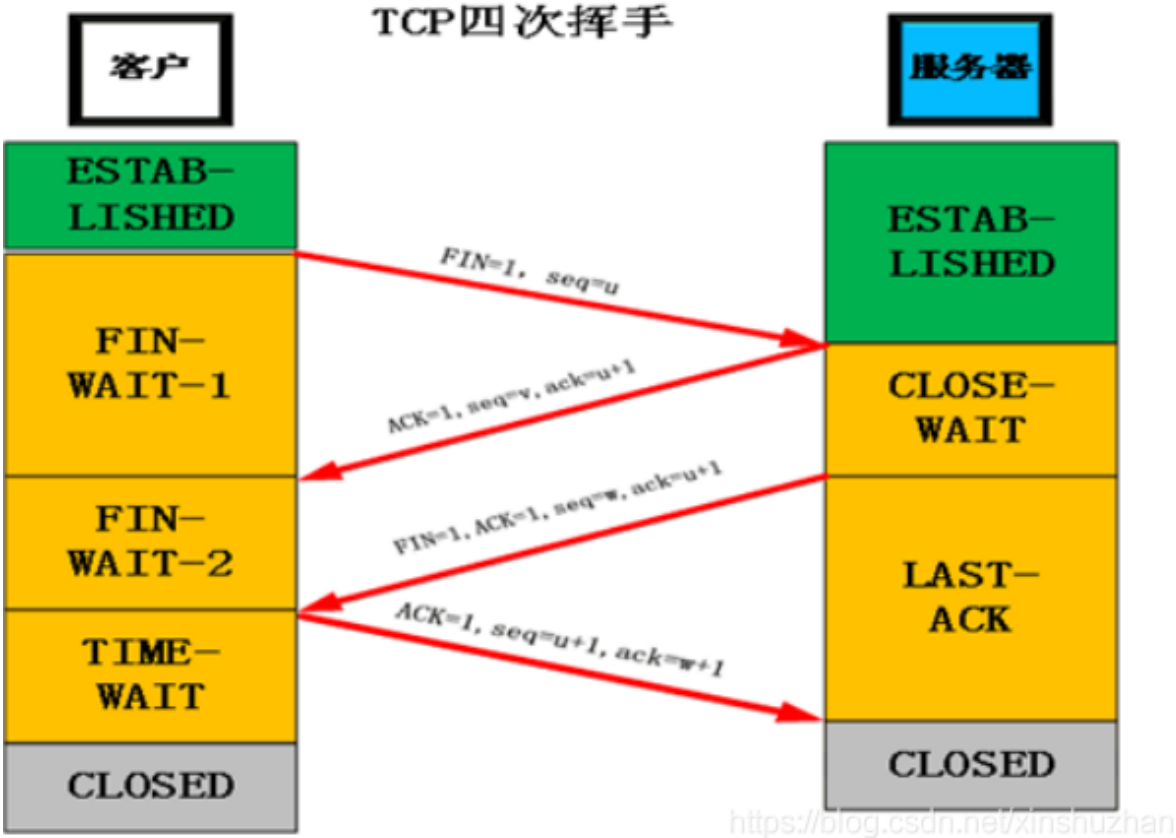
并且再次回复ACK验证请求，在B端发送过来的seq基础上加1，进行回复；同时也会回复ack确认控制字段，

以便B收到信息时，知晓自己的TCP建立请求已经得到了确认。



四次挥手
第一次挥手：
Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN_WAIT_1状态。
第二次挥手：
Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE_WAIT状态。
第三次挥手：
Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST_ACK状态。
第四次挥手：
Client收到FIN后，Client进入TIME_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，

Server进入CLOSED状态，完成四次挥手。



52. 什么是动态资源，什么是静态资源

静态资源：可以理解为前端的固定页面，这里面包含HTML、CSS、JS、图片等等，不需要查数据库也不需要程序处理，直接就能够显示的页面，如果想修改内容则必须修改页面，但是访问效率相当高。

动态资源：一般客户端请求的动态资源，先将请求交于web容器，web容器连接数据库，数据库处理数据之后，将内容交给web服务器，web服务器返回给客户端解析渲染处理。

53. worker支持的最大并发数是什么?

worker的连接数是2个时，最大并发数：(worker_connection * worker_processes) /2；worker的连接数是4个时，最大并发数：(worker_connection * worker_processes) /4

54. Tomcat和Resin有什么区别，工作中你怎么选择?

区别：Tomcat用户数多，可参考文档多，Resin用户数少，可考虑文档少
最主要区别则是Tomcat是标准的java容器，不过性能方面比resin的要差一些
但稳定性和java程序的兼容性，应该是比resin的要好

工作中选择：现在大公司都是用resin，追求性能；而中小型公司都是用Tomcat，追求稳定和程序的兼容

55. 什么叫网站灰度发布?

也叫金丝雀发布
AB test就是一种灰度发布方式，让一部用户继续用A，一部分用户开始用B
如果用户对B没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到B上面 来
灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度

56.. 统计ip访问情况，要求分析nginx访问日志，找出访问页面数量在前十位的ip

cat access.log | awk '{print $1}' | uniq -c | sort -rn | head -10

57. nginx各个版本的区别

Nginx官网提供了三个类型的版本
Mainline version：Mainline 是 Nginx 目前主力在做的版本，可以说是开发版
Stable version：最新稳定版，生产环境上建议使用的版本
Legacy versions：遗留的老版本的稳定版

58. nginx最新版本

1.19，稳定版本1.18

59. 关于nginx access模块的面试题

编写一个Nginx的access模块，要求准许192.168.3.29/24的机器访问，准许10.1.20.6/16这个网段的所有机器访问，准许34.26.157.0/24这个网段访问,除此之外的机器不准许访问。

```
location/{

access 192.168.3.29/24;

access 10.1.20.6/16;

access 34.26.157.0/24;

deny all;

}
```

60. nginx默认配置文件

在 nginx 的配置文件中，大概分为几个区域：events {}、http {}、和没有被 {}包裹的区域。而 http {} 中还有 server {}，以及 server {} 中的 location {}。结构如下：

```
...
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    ...

    server {
        ...

        location {
            ...
        }
    }
```

```
    server {
        ...
    }
}
```

- 没有被 {} 包裹的部分为全局配置，如 worker_processes 1; 设置工作进程（子进程）数为 1
- events {} 为 nginx 连接配置的模块，如 worker_connections 1024; 设置每一个子进程最大允许连接 1024 个连接
- http {} 为 nginx http 核心配置模块
- server {} 为虚拟主机配置模块，包括监听端口、监听域名等
- location {} URI 匹配

61. location的规则

在 Nginx 的配置文件中，通常会用两个常用的区块(Block)来进行设置：

1.Server 区块

2.Localtion 区块

sever 区块主要是真的主机的配置，比如配置主机的域名，IP，端口等内容。当然，在一个 Nginx 的配置文件里面，我们是可以指定多个 Sever 区块的配置的。
而 Location 区块则是在 Sever 区块里面，细分到针对不同的路径和请求而进行的配置。因为一个站点中的 URI 通常会非常多，所以在 Location 区块设置这部分，你也是可以写多个 Location 的配置的。

```
location optional_modifier location_match {
 # 这个 {} 里面的配置内容就是一个区块 Block
}
```

上面的 optional_modifier 配置项是可以使用正则表达式的。常用的几种如下:

```
留空。在留空的情况下，配置表示请求路径由 location_match 开始。
= ，等于号还是非常容易理解的：就是请求路径正好等于后面的 location_match 的值；跟第一项留空还是有区别的。
~，飘号（注意是英文输入的飘号）表示大小写敏感的正则匹配。
~*表示大小写不敏感的正则匹配。
^~ 表示这里不希望有正则匹配发生。
```

nginx 处理localtion区块的顺序

```
每一个请求进来 Nginx 之后，Nginx 就会选择一个 Location 的最佳匹配项进行响应，处理的具体流程是逐一跟 location 的配置进行比对，这个步骤可以分为以下几步：

先进行前缀式的匹配（也就是 location 的 optional_modifier 为空的配置）。
```

```
Nginx 其次会根据 URI 寻找完全匹配的 location 配置（也就是 location 的 optional_modifier 为 = 的配置）.
如果还是没有匹配到，那就先匹配 ^~ 配置，如果找到一个配置的话，则会停止寻找过程，直接返回响应内容。
如果还是没有找到匹配项的话，则会先进行大小写敏感的正则匹配，然后再是大小不写敏感的正则匹配
```

举例子:

```
location = / {
 # = 等号配置符，只匹配 / 这个路由
}

location /data {
  # 留空配置，会匹配有 /data 开始的路由，后续有匹配会往下匹配。
}

location ^~ /img/ {
 # 注意 ^~ 配置，这里匹配到 /img/ 开始的话，直接就返回了。
}

location ~* .(png|gif|ico|jpg|jpeg)$ {
 # 匹配以 png, gif, ico, jpg or jpeg 结尾的请求；这个通常用来设置图片的请求响应。
}
```

62. 配置nginx防盗链

Nginx的防盗链原理是加入location项，用正则表达式过渡图片类型文件，对于信任的网址可以正常使用，对于不信任的网址则返回相应的错误图片，在源主机（bt.com）的配置文件中加入以下代码：

```
vi /usr/local/nginx/conf/nginx.conf
location ~*\.(jpg|gif|swf)$ {
valid_referers none blocked *.test.com test.com;
if ($invalid_referer) {
rewrite ^/http://www.bt.com/error.png;
}
}
```

下面分析一下这段代码：
~*\.(jpg|gif|swf)$：这段正则表达式表示匹配不区分大小写，以.jpg或.gif或.swf结尾的文件。
valid_referers：设置信任的网站，可以正常使用图片。
none：浏览器中referer为空的情况，这就是直接在浏览器访问图片。
blocked：浏览器中referer不可空的情况，但是值被代理或防火墙删除了，这些值不以http://或https://开头。
后面的网站或者域名：referer中包含相关字符串的网址。
if语句：如果链接的来源域名不在valid_referers所列出的列表中，$invalid_referer为1，则执行后面的操作，即进行重写或返回403页面。

把图片error.png放到源主机（bt.com）的工作目录下。

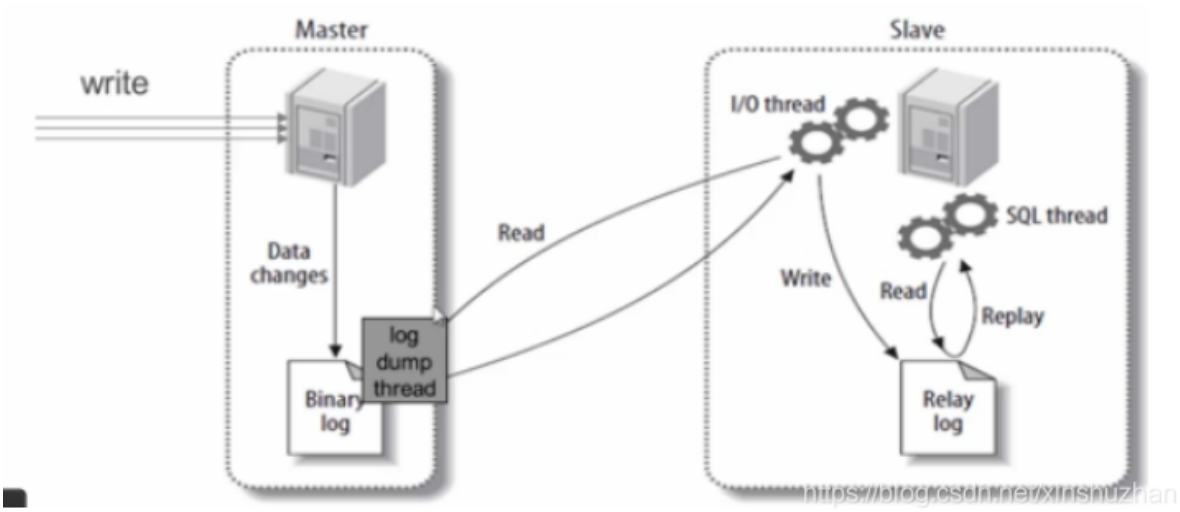
```
ls /usr/local/nginx/html
50x.html index.html logo.jpg error.png
```

这是重启服务器，重新访问http://www.test.com/index.html，显示的是被重写的图片。

63. drop，delete和truncate删除数据的区别?

- delete 语句执行删除是每次从表中删除一行，并且同时将改行的删除操作作为事务记录在日志中保存以便进行回滚。
- truncate 则是一次从表中删除所有的数据并不把单独的删除操作记录计入日志，删除行是不能恢复的。执行速度很快
- drop 是将表所占的空间全部释放掉。
  在删除速度上， drop>truncate>delete
  想要删除部分数据用delete，想要删除表用drop。 想保留表但是把数据删除，如果和事务无关用truncate

64. MySQL主从原理



从库生成两个线程，一个I/O线程，一个SQL线程；

i/o线程去请求主库 的binlog，并将得到的binlog日志写到relay log(中继日志) 文件中；

主库会生成一个 log dump 线程，用来给从库 i/o线程传binlog；

SQL 线程，会读取relay log文件中的日志，并解析成具体操作，来实现主从的操作一致，而最终数据一致；

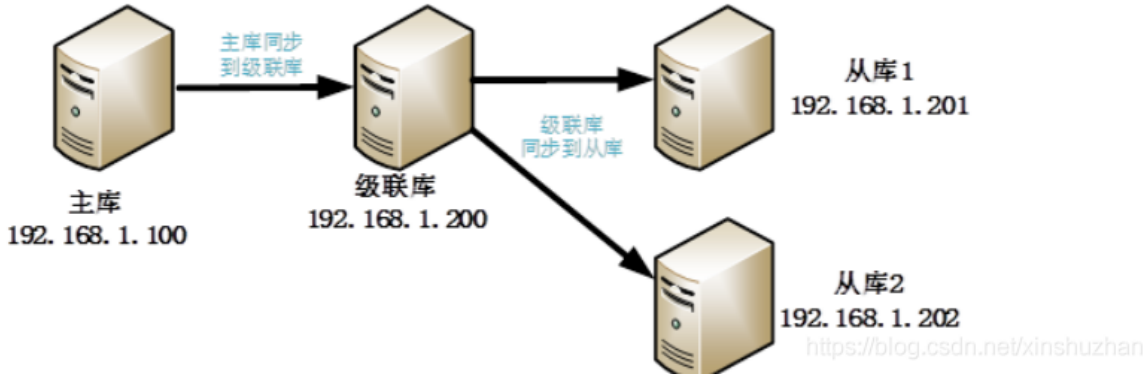
65. MySQL主从复制存在哪些问题?

mysql主从复制存在的问题：主库宕机后，数据可能丢失，从库只有一个sql Thread，主库写压力大，复制很可能延时。

解决方法： 用半同步复制解决数据丢失的问题
用并行复制解决从库复制延迟的问题。

66. MySQL复制的方法

级联复制：将主库的数据同步到级联库，然后级联库把自己的数据同步到从库上，这样可以减少主库的压力



半同步复制：
默认情况下，MySQL的复制功能是异步的，异步复制可以提供最佳的性能，主库把binlog日志发送给从库即结束，并不验证从库是否接收完毕。这意味着当主库或从库发生故障时，有可能从库没有接收到主库发送过来的binlog日志，这就会造成主库和从库的数据不一致，甚至在恢复时造成数据的丢失。

在开启了半同步复制机制后，主库只有当有任意一台从库已经接收到主库的数据后，告诉主库。主库收到从库同步成功的信息后，才继续后面的操作。

67. 主从延迟产生的原因及解决方案?

- 主库的并发比较高的时候，产生的DDL数量超过了从库的一个sql线程所承受的范围，那么延时就产生了。
- 还有可能是与从库的大型query语句产生的了锁等待 。
- 网络抖动

解决方案：
1)、架构方面

1.业务的持久化层的实现采用分库架构，mysql服务可平行扩展，分散压力。

2.单个库读写分离，一主多从，主写从读，分散压力。这样从库压力比主库高，保护主库。

3.服务的基础架构在业务和mysql之间加入memcache或者redis的cache层。降低mysql的读压力。

4.不同业务的mysql物理上放在不同机器，分散压力。

5.使用比主库更好的硬件设备作为slave总结，mysql压力小，延迟自然会变小。

2)、mysql主从同步加速

1、sync_binlog在slave端设置为0

2、–logs-slave-updates 从服务器从主服务器接收到的更新不记入它的二进制日志。

3、直接禁用slave端的binlog

68. 判断主从延迟的方法

可以通过命令 show slave status 查看

比如通过seconds_behind_master的值来判断
NULL - 表示io_thread或是sql_thread有任何一个发生故障，也就是该线程的Running状态是No,而非Yes.
0 - 该值为零，是我们极为渴望看到的情况，表示主从复制状态正常

69. MySQL忘记root密码如何找回

1. 在配置文件里加上skip-grant-tables ， 重启MySQL
2. 使用MySQL-uroot -p 进入
3. 使用update 修改密码

```
mysql> USE mysql ;
mysql> UPDATE user SET Password = password ( 'new-password' ) WHERE User =
'root' ;
```

70. MySQL的数据备份方式

工具一 MySQLdump工具备份
工具二： xtrabackup工具备份

备份分为：冷备，温备和热备

根据要备份的数据集合又分为： 完全备份，增量备份和差异备份

需要备份的对象：

- 数据
- 配置文件

- OS相关的配置文件
- 代码： 存储过程，存储函数和处罚器
- 复制相关的配置
- 二进制日志

数据量比较大的时候用xtrabackup

基于MySQLdump做备份策略： 周日做全备，备份同时滚动日志
周一到周六：备份二进制文件

恢复的时候： 完全备份+二进制文件中到此处的事件

xtrabackup的特点：

```
1）备份过程快速、可靠；

2）备份过程不会打断正在执行的事务；

3）能够基于压缩等功能节约磁盘空间和流量；

4）自动实现备份检验；

5）还原速度快；
```

特性\工具	xtrabackup	mydumper	MySQLdump
性能	最佳	次之	最差
数据量	最大	最少	次之
远程备份	不可以	可以	可以
数据一致性	支持	支持	支持
工具便捷性	次之	最佳	最差
MySQL 版本兼容性	较差	较好	较好
备份类型	物理备份	逻辑备份	逻辑备份

逻辑备份： 类似于select * from 查询满足条件的备份
物理本分： 备份文件+日志文件
所以xtrabackup就是物理备份
MySQLdump就是逻辑备份

71. innodb的特性

一：插入缓冲

二：二次写

三：自适应哈希

四：预读

72. varchar(100) 和varchar(200）的区别

varchar(100)最多存放100个字符，varchar(200)最多存放200个字符，varchar(100)和(200)存储hello所占空间一样，但后者在排序时会消耗更多内存，因为order by col采用fixed_length计算col长度(memory引擎也一样

73. MySQL主要的索引类型

普通索引：是最基本的索引，它没有任何限制；

唯一索引：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一；

主键索引：是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值；

组合索引：指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用组合索引时遵循最左前缀集合；

全文索引：主要用来查找文本中的关键字，而不是直接与索引中的值相比较，mysql中MyISAM支持全文索引而InnoDB不支持；

74. 请说出非关系型数据库的典型产品、特点及应用场景?

MongoDB
特点：1.高性能，易部署，易使用。
2.面向集合存储，易存储对象类型的数据。
3.模式自由
4.自动处理碎片，以支持云计算层次的扩展性。
应用场景：
网站数据：mongodb非常适合实时的插入，更新与查询。
缓存：适合作为信息基础设施的缓存层
大尺寸、低价值的数据
高伸缩性的场景
Redis
特点：1.性能极高，能支持超过100k+每秒的读写频率
2.丰富的数据类型
3.所有操作都是原子性的
使用场景：
少量的数据存储，高速读写访问
SQLlite
特点：
1.嵌入式的，零配置，无需安装和管理配置
2.ACID事务
3.存储在单一磁盘文件中的一个完整的数据库。
应用场景：
1.需要数据库的小型桌面软件。
2.需要数据库的手机软件。
3.作为数据容器的应用场景。

75. 如何加强MySQL安全，请给出可行的具体措施?

1.避免直接从互联网访问mysql数据库，确保特定主机才拥有访问权限。
2.定期备份数据库
3.禁用或限制远程访问
在my.cnf文件里设置bind-address指定ip
4.移除test数据库（默认匿名用户可以访问test数据库）

5.禁用local infile
mysql> select load_file("/etc/passwd");
在my.cnf里[mysqld]下添加set-variable=local-infile=0
6.移除匿名账户和废弃的账户
7.限制mysql数据库用户的权限
8.移除和禁用.mysql_history文件

76. Binlog工作模式有哪些？各什么特点，企业如何选择？

1.row level行级模式
优点：记录数据详细（每行），主从一致
缺点：占用大量的磁盘空间，降低了磁盘的性能
2.statement level模式（默认）
优点：记录的简单，内容少，节约了IO，提高性能　缺点：导致主从不一致
3.MIXED混合模式
结合了statement和row模式的优点，会根据执行的每一条具体的SQL语句来区分对待记录的日志形式。
对于函数，触发器，存储过程会自动使用row level模式
企业场景选择：
1.互联网公司使用mysql的功能较少（不用存储过程、触发器、函数），选择默认的statement模式。
2.用到mysql的特殊功能（存储过程、触发器、函数）则选则MIXED模式
3.用到mysql的特殊功能（存储过程、触发器、函数），有希望数据最大化一致则选择row模式。

77. 生产一主多从从库宕机，如何手工恢复？

处理方法：重做slave

1. 停止slave
2. 导入备份数据
3. 配置master.info信息
4. 启动slave
5. 检查从库状态

78. MySQL中MyISAM与InnoDB的区别，至少5点

a. InnoDB支持事务，而MyISAM不支持事务。
b. InnoDB支持行级锁，而MyISAM支持表级锁
c. InnoDB支持MVCC，而MyISAM不支持
d. InnoDB支持外键，而MyISAM不支持
e. InnoDB不支持全文索引，而MyISAM支持

79. 网站打开慢，请给出排查方法，如是数据库慢导致，如何排查并解决，请分析并举例？

1. 检查操作系统是否负载过高
2. 登陆mysql查看有哪些sql语句占用时间过长，show processlist;
3. 用explain查看消耗时间过长的SQL语句是否走了索引
4. 对SQL语句优化，建立索引

80. xtrabackup的备份，增量备份及恢复的工作原理

XtraBackup基于InnoDB的crash-recovery功能，它会复制InnoDB的data file，由于不锁表，复制出来的数据是不一致的，在恢复的时候使用crash-recovery，使得数据恢复一致。
InnoDB维护了一个redo log，又称为transaction log（事务日志），它包含了InnoDB数据的所有改动情况。当InnoDB启动的时候，它会先去检查data file和transaction log，并且会做两步操作：
XtraBackup在备份的时候，一页一页的复制InnoDB的数据，而且不锁定表，与此同时，XtraBackup还有另外一个线程监视着transaction log，一旦log发生变化，就把变化过的log pages复制走。为什么要着急复制走呢？因为transaction log文件大小有限，写满之后，就会从头再开始写，所以新数据可能会覆盖到旧的数据。
在prepare过程中，XtraBackup使用复制到的transaction log对备份出来的InnoDB data file进行crash recover

81.误执行drop数据，如何通过xtrabackup恢复?

1. 关闭mysql服务
2. 移除mysql的data目录及数据
3. 将备份的数据恢复到mysql的data目录
4. 启动mysql服务

82. 如何做主从数据一致性校验?

主从一致性校验有多种工具 例如checksum、mysqldiff、pt-table-checksum等

83. MySQL有多少日志

错误日志：记录出错信息，也记录一些警告信息或者正确的信息。

查询日志：记录所有对数据库请求的信息，不论这些请求是否得到了正确的执行。

慢查询日志：设置一个阈值，将运行时间超过该值的所有SQL语句都记录到慢查询的日志文件中。

二进制日志：记录对数据库执行更改的所有操作。

中继日志。

事务日志。
le/details/111957929

84. MySQL binlog的几种日志录入格式以及区别

1.Statement：每一条会修改数据的sql都会记录在binlog中。
优点：不需要记录每一行的变化，减少了binlog日志量，节约了IO，提高性能。(相比row能节约多少性能 与日志量，这个取决于应用的SQL情况，正常同一条记录修改或者插入row格式所产生的日志量还小于Statement产生的日志量，但是考虑到如果带条 件的update操作，以及整表删除，alter表等操作，ROW格式会产生大量日志，因此在考虑是否使用ROW格式日志时应该跟据应用的实际情况，其所 产生的日志量会增加多少，以及带来的IO性能问题。)
缺点：由于记录的只是执行语句，为了这些语句能在slave上正确运行，因此还必须记录每条语句在执行的时候的 一些相关信息，以保证所有语句能在slave得到和在master端执行时候相同 的结果。另外mysql 的复制,像一些特定函数功能，slave可与master上要保持一致会有很多相关问题(如sleep()函数，last_insert_id()，以及user-defined functions(udf)会出现问题).
使用以下函数的语句也无法被复制：

- LOAD_FILE()
- UUID()
- USER()

- FOUND_ROWS()
- SYSDATE() (除非启动时启用了 --sysdate-is-now 选项)
  同时在INSERT ...SELECT 会产生比 RBR 更多的行级锁

2.Row:不记录sql语句上下文相关信息，仅保存哪条记录被修改。
优点： binlog中可以不记录执行的sql语句的上下文相关的信息，仅需要记录那一条记录被修改成什么了。所以rowlevel的日志内容会非常清楚的记录下 每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或function，以及trigger的调用和触发无法被正确复制的问题
缺点:所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容,比 如一条update语句，修改多条记录，则binlog中每一条修改都会有记录，这样造成binlog日志量会很大，特别是当执行alter table之类的语句的时候，由于表结构修改，每条记录都发生改变，那么该表每一条记录都会记录到日志中。

3.Mixedlevel: 是以上两种level的混合使用，一般的语句修改使用statment格式保存binlog，如一些函数，statement无法完成主从复制的操作，则 采用row格式保存binlog,MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择 一种.新版本的MySQL中队row level模式也被做了优化，并不是所有的修改都会以row level来记录，像遇到表结构变更的时候就会以statement模式来记录。至于update或者delete等修改数据的语句，还是会记录所有行的变更。

85. MySQL数据库cpu飙升到500%的话他怎么处理?

当 cpu 飙升到 500%时，先用操作系统命令 top 命令观察是不是 mysqld 占用导致的，如果不是，找出占用高的进程，并进行相关处理

如果是 mysqld 造成的， show processlist，看看里面跑的 session 情况，是不是有消耗资源的 sql 在运行。找出消耗高的 sql，
看看执行计划是否准确， index 是否缺失，或者实在是数据量太大造成。

一般来说，肯定要 kill 掉这些线程(同时观察 cpu 使用率是否下降)，等进行相应的调整(比如说加索引、改 sql、改内存参数)之后，再重新跑这些 SQL。

也有可能是每个 sql 消耗资源并不多，但是突然之间，
有大量的 session 连进来导致 cpu 飙升，这种情况就需要跟应用一起来分析为何连接数会激增，再做出相应的调整，比如说限制连接数等

86. redis是单线程还是多线程?

这个问题已经被问过很多次了，从redis4.0开始引入多线程，redis 6.0 中，多线程主要用于网络 I/O 阶段，也就是接收命令和写回结果阶段，而在执行命令阶段，还是由单线程串行执行。由于执行时还是串行，因此无需考虑并发安全问题。
redis 中的多线程组不会同时存在“读”和“写”，这个多线程组只会同时“读”或者同时“写”
在 redis 6.0 之前，redis 的核心操作是单线程的。

因为 redis 是完全基于内存操作的，通常情况下CPU不会是redis的瓶颈，redis 的瓶颈最有可能是机器内存的大小或者网络带宽。

既然CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了，因为如果使用多线程的话会更复杂，同时需要引入上下文切换、加锁等等，会带来额外的性能消耗。

而随着近些年互联网的不断发展，大家对于缓存的性能要求也越来越高了，因此 redis 也开始在逐渐往多线程方向发展。

87. redis常用的版本是?

redis5.0，redis6.0

88. redis 的使用场景?

缓存、分布式锁、排行榜（zset）、计数（incrby）、消息队列（stream）、地理位置（geo）、访客统计（hyperloglog）

89. redis常见的数据结构

常见的5种：

String：字符串，最基础的数据类型。

List：列表。

Hash：哈希对象。

Set：集合。

Sorted Set：有序集合，Set 的基础上加了个分值。

高级的2 种：

- HyperLogLog：通常用于基数统计。使用少量固定大小的内存，来统计集合中唯一元素的数量。统计结果不是精确值，而是一个带有0.81%标准差（standard error）的近似值。所以，HyperLogLog适用于一些对于统计结果精确度要求不是特别高的场景，例如网站的UV统计。
- Stream：主要用于消息队列，类似于 kafka，可以认为是 pub/sub 的改进版。提供了消息的持久化和主备复制功能，可以让任何客户端访问任何时刻的数据，并且能记住每一个客户端的访问位置，还能保证消息不丢失。

90. redis持久化你们怎么做的?

redis持久化主要有两种 ROD和AOF，当先现在还有混合的，从reids4.0后引入的

RDB实现原理：
RDB类似于快照，在某个时间点，将 Redis 在内存中的数据库状态（数据库的键值对等信息）保存到磁盘里面。RDB 持久化功能生成的 RDB 文件是经过压缩的二进制文件。

RDB的优点：

- RDB文件是经过压缩的，占用空间很小，它保存了某个时间点的数据集，很适合做备份。 比如你可以在24小时内，每个小时备份一次RDB文件，并且每个月的每一天备份一个RDB文件。
- RDB非常适合用来做灾备恢复，可以加密后传送到数据中心
- RDB可以最大化redis的性能
- 从恢复速度来看，RDB明显要比AOF要快
  但是RDB也有一定的缺点：
- RDB在服务器故障的时候，容易造成数据损失。 我们通常设置每5分钟保存一次快照，这样数据丢失也只有5分钟的数据。
- RDB保存时使用fork子进程数据的持久化，如果数据量大的话，会非常耗时，造成redis停止处理服务N毫秒。

AOF：
保存 Redis 服务器所执行的所有写操作命令来记录数据库状态，并在服务器启动时，通过重新执行这些命令来还原数据集。

AOF默认是关闭的，可以通过appendonley yes 开启

AOF 持久化功能的实现可以分为三个步骤：命令追加、文件写入、文件同步。

AOF的优点：
1）AOF 比 RDB可靠。你可以设置不同的 fsync 策略：no、everysec 和 always。默认是 everysec，在这种配置下，redis 仍然可以保持良好的性能，并且就算发生故障停机，也最多只会丢失一秒钟的数据。

2）AOF文件是一个纯追加的日志文件。即使日志因为某些原因而包含了未写入完整的命令（比如写入时磁盘已满，写入中途停机等等）， 我们也可以使用 redis-check-aof 工具也可以轻易地修复这种问题。

3）当 AOF文件太大时，Redis 会自动在后台进行重写：重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。整个重写是绝对安全，因为重写是在一个新的文件上进行，同时 Redis 会继续往旧的文件追加数据。当新文件重写完毕，Redis 会把新旧文件进行切换，然后开始把数据写到新文件上。

4） AOF 文件有序地保存了对数据库执行的所有写入操作以 Redis 协议的格式保存， 因此 AOF 文件的内容非常容易被人读懂， 对文件进行分析（parse）也很轻松。如果你不小心执行了 FLUSHALL 命令把所有数据刷掉了，但只要 AOF 文件没有被重写，那么只要停止服务器， 移除 AOF 文件末尾的 FLUSHALL 命令， 并重启 Redis ， 就可以将数据集恢复到 FLUSHALL 执行之前的状态。

AOF缺点：
1） 对于相同的数据集，AOF的文件一般会比RDB大
2） AOF所使用的fsync策略，备份速度也会比RDB曼

如何使用：
如果想尽量保证数据安全性， 你应该同时使用 RDB 和 AOF 持久化功能，同时可以开启混合持久化。
如果想尽量保证数据安全性， 你应该同时使用 RDB 和 AOF 持久化功能，同时可以开启混合持久化。
如果你的数据是可以丢失的，则可以关闭持久化功能，在这种情况下，Redis 的性能是最高的。

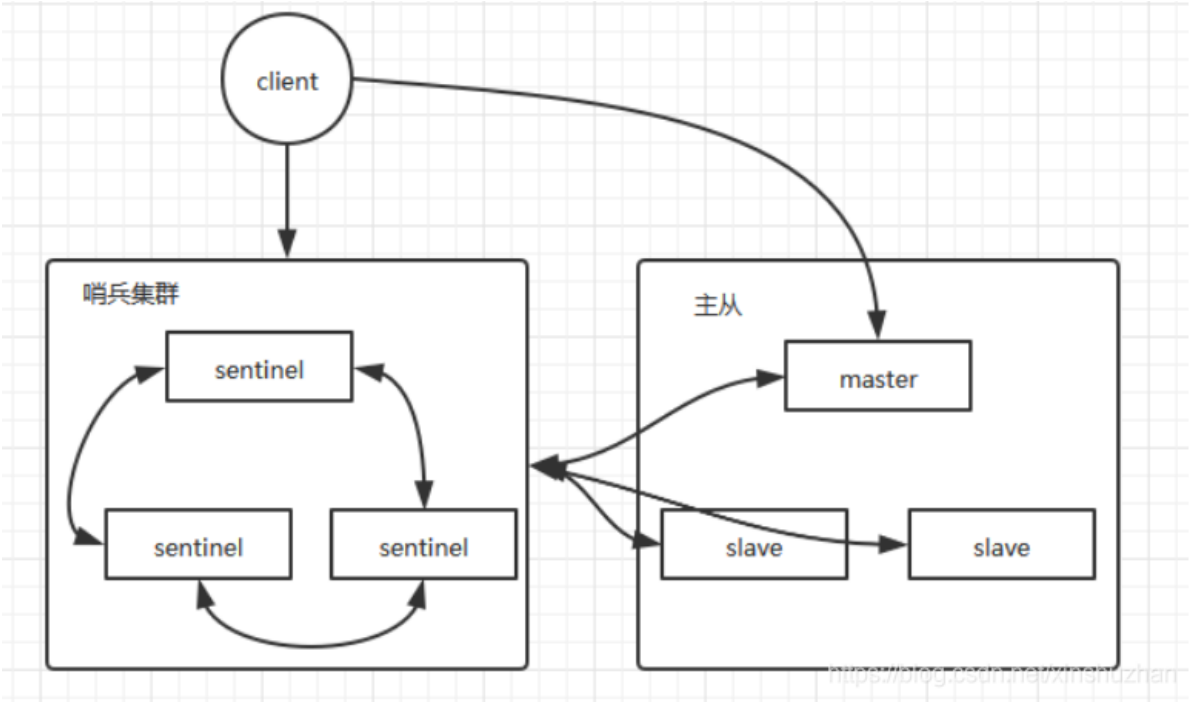
91. 主从复制实现的原理

Redis虽然读取写入的速度都特别快，但是也会产生读压力特别大的情况，为分担读压力，Redis支持主从复制，Redis的主从结构可以采用一主多从或者级联结构，Redis主从复制可以根据是否是全量分为全量同步和增量同步

92. redis哨兵模式原理

哨兵是特殊的redis服务，不提供读写服务，主要用来监控redis实例节点。 哨兵架构下client端第一次从哨兵找出redis的主节点，后续就直接访问redis的主节点，不会每次都通过 sentinel代理访问redis的主节点，当redis的主节点发生变化，哨兵会第一时间感知到，并且哨兵会早主从模式的从节点中重新选出来一个新的master，并且将新的master信息通知给client端。

这里面redis的client端一般都实现了订阅功能，订阅sentinel发布的节点变动消息。Redis服务是通过配置文件启动的，比如上面的从节点设置了只读模式，它被选举成了master之后就是可读写的了，感觉很奇怪，后来看了下重新选举之后的各redis服务的配置文件，发现文件里面的内容会被哨兵修改。要想真的高可用，我们的哨兵也要集群模式。



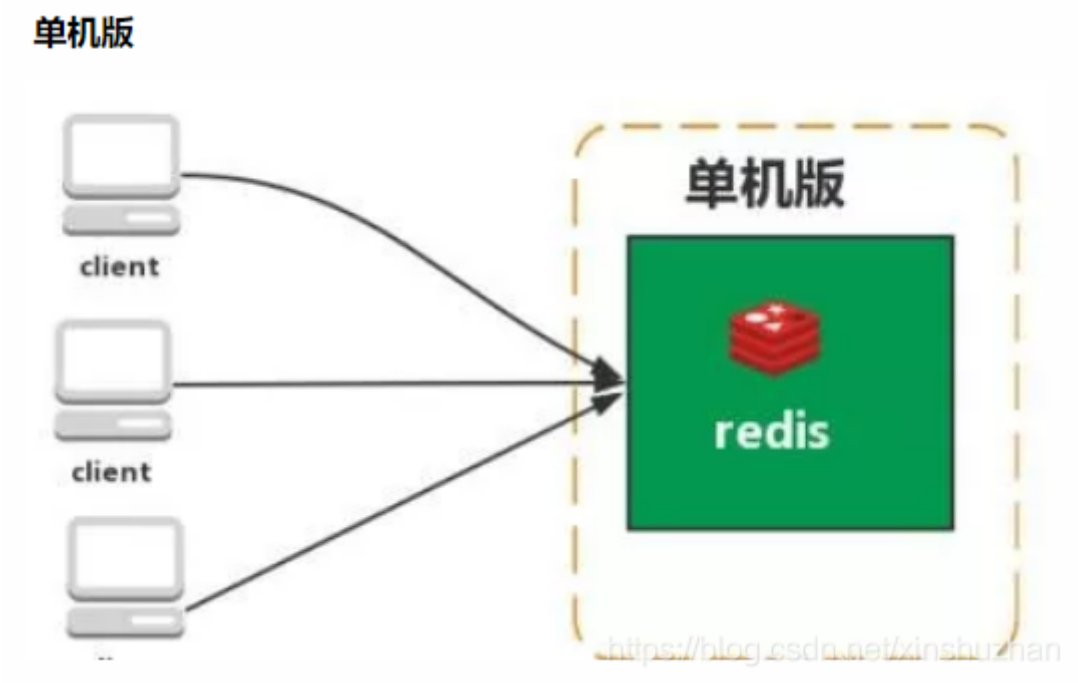
93. memcache和redis的区别

1、 Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等。
2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储。
3、虚拟内存–Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘
4、过期策略–memcache在set时就指定，例如set key1 0 0 8,即永不过期。Redis可以通过例如expire设定，例如expire name 10
5、分布式–设定memcache集群，利用magent做一主多从;redis可以做一主多从。都可以一主一从
6、存储数据安全–memcache挂掉后，数据没了；redis可以定期保存到磁盘（持久化）
7、灾难恢复–memcache挂掉后，数据不可恢复; redis数据丢失后可以通过aof恢复
8、Redis支持数据的备份，即master-slave模式的数据备份。

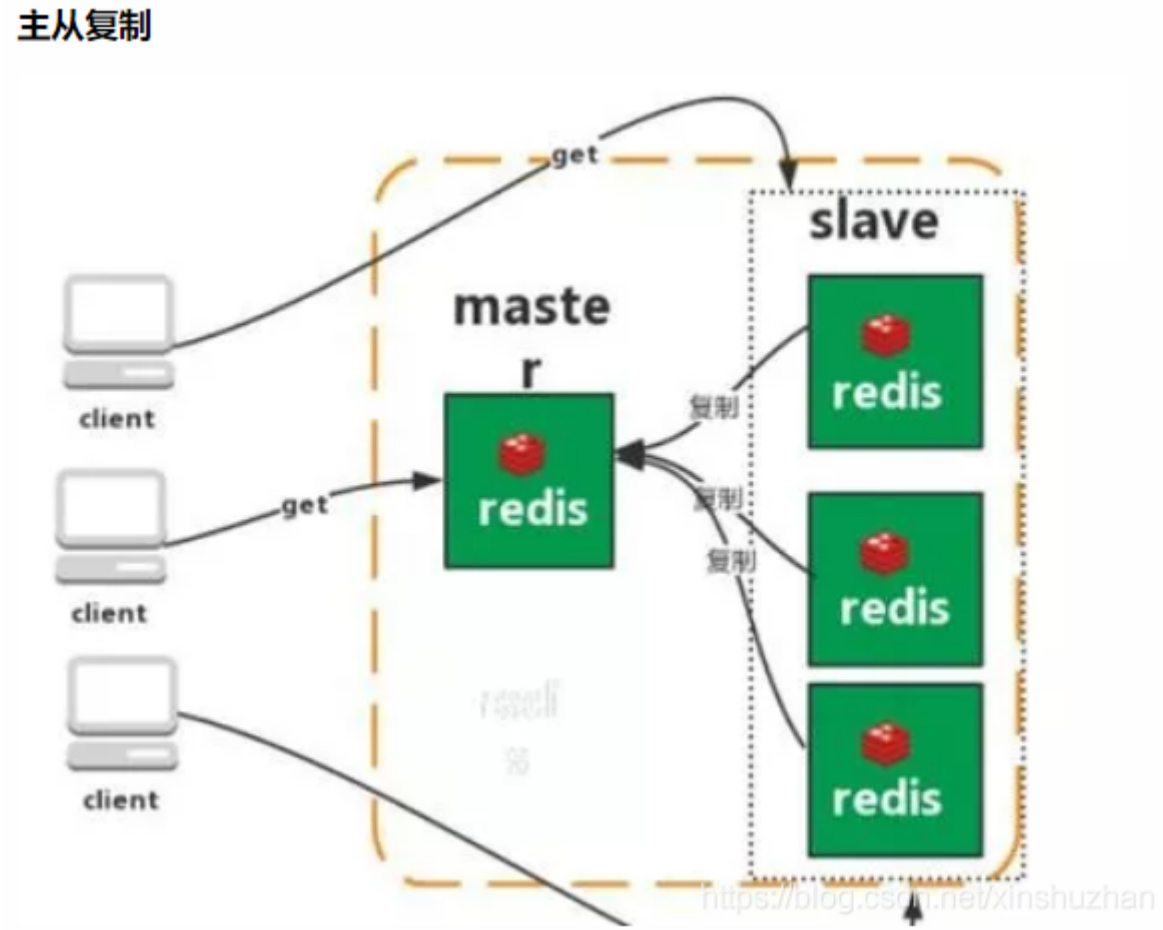
redis和memecache的不同在于[2]：
1、存储方式：
memecache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小
redis有部份存在硬盘上，这样能保证数据的持久性，支持数据的持久化（笔者注：有快照和AOF日志两种持久化方式，在实际应用的时候，要特别注意配置文件快照参数，要不就很有可能服务器频繁满载做dump）。
2、数据支持类型：
redis在数据支持上要比memecache多的多。
3、使用底层模型不同：
新版本的redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。
4、运行环境不同：
redis目前官方只支持LINUX 上去行，从而省去了对于其它系统的支持，这样的话可以更好的把精力用于本系统 环境上的优化，虽然后来微软有一个小组为其写了补丁。但是没有放到主干上

个人总结一下，有持久化需求或者对数据结构和处理有高级要求的应用，选择redis，其他简单的key/value存储，选择memcache。

94. redis有哪些架构模式?



存在问题： 内容容量有限，处理能力有限，无法高可用



Redis 的复制（replication）功能允许用户根据一个 Redis 服务器来创建任意多个该服务器的复制品，其中被复制的服务器为主服务器（master），而通过复制创建出来的服务器复制品则为从服务器（slave）。 只要主从服务器之间的网络连接正常，主从服务器两者会具有相同的数据，主服务器就会一直将发生在自己身上的数据更新同步 给从服务器，从而一直保证主从服务器的数据相同。

特点：

1、master/slave 角色

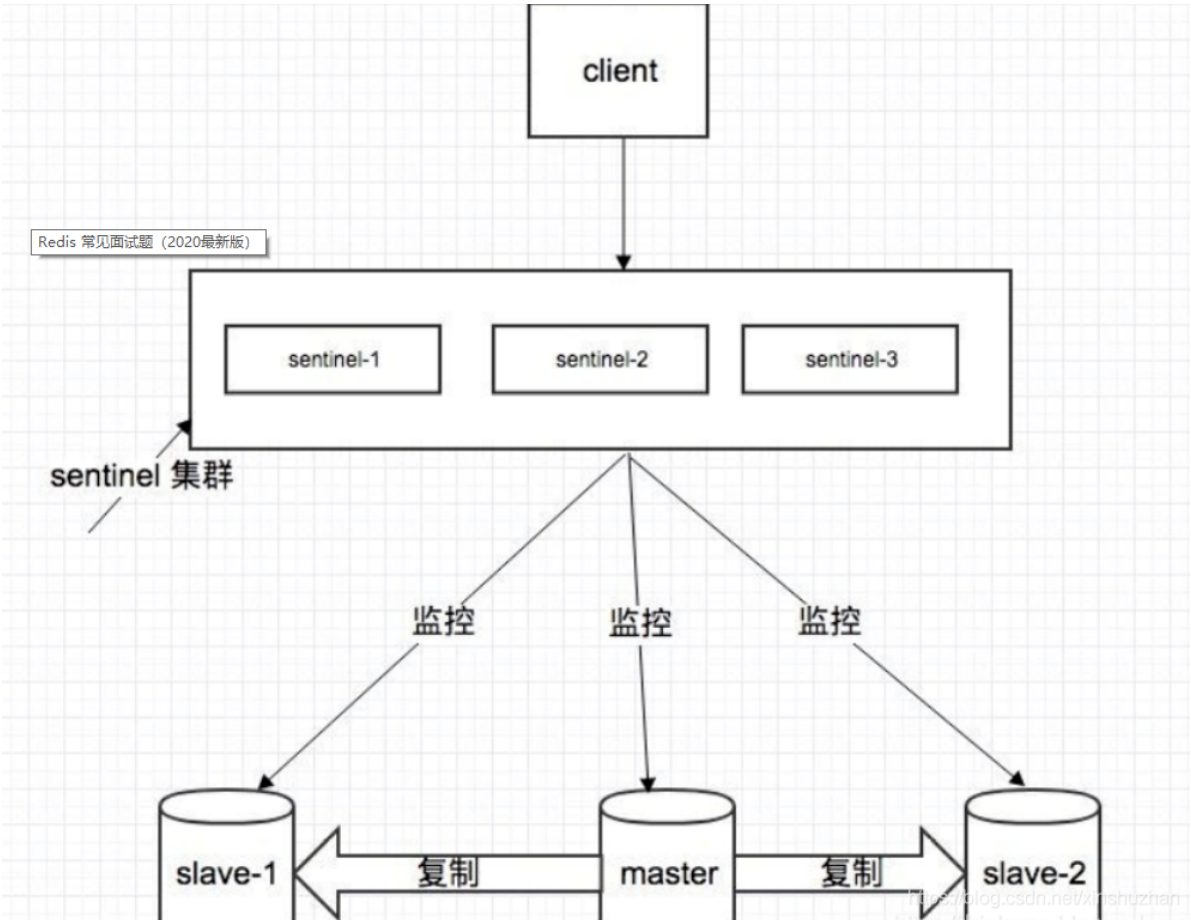
2、master/slave 数据相同

3、降低 master 读压力在转交从库

问题：

无法保证高可用

没有解决 master 写的压力



Redis sentinel 是一个分布式系统中监控 redis 主从服务器，并在主服务器下线时自动进行故障转移。其中三个特性：

监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作。

特点：

1、保证高可用

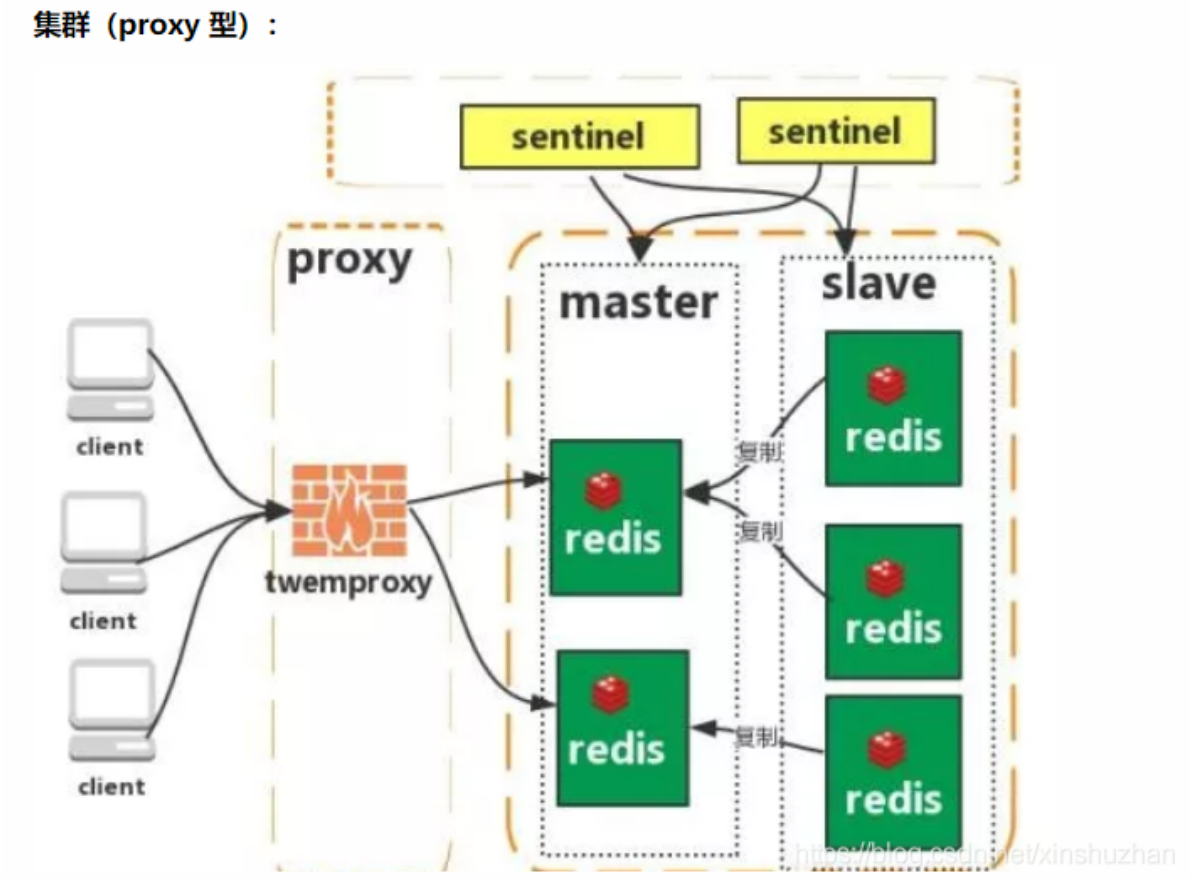
2、监控各个节点

3、自动故障迁移

缺点：主从模式，切换需要时间丢数据

没有解决 master 写的压力

集群（proxy 型）：



Twemproxy 是一个 Twitter 开源的一个 redis 和 memcache 快速/轻量级代理服务器； Twemproxy 是一个快速的单线程代理程序，支持 Memcached ASCII 协议和 redis 协议。

特点：1、多种 hash 算法：MD5、CRC16、CRC32、CRC32a、hsieh、murmur、Jenkins

2、支持失败节点自动删除

3、后端 Sharding 分片逻辑对业务透明，业务方的读写方式和操作单个 Redis 一致

缺点：增加了新的 proxy，需要维护其高可用。

failover 逻辑需要自己实现，其本身不能支持故障的自动转移可扩展性差，进行扩缩容都需要手动干预

95. 缓存雪崩?

在前面学习我们都知道Redis不可能把所有的数据都缓存起来(内存昂贵且有限)，所以Redis需要对数据设置过期时间，并采用的是惰性删除+定期删除两种策略对过期键删除。Redis对过期键的策略+持久化

如果缓存数据设置的过期时间是相同的，并且Redis恰好将这部分数据全部删光了。这就会导致在这段时间内，这些缓存同时失效，全部请求到数据库中

什么是缓存雪崩?
Redis挂掉了，请求全部走数据库。

对缓存数据设置相同的过期时间，导致某段时间内缓存失效，请求全部走数据库。

缓存雪崩如果发生了，很可能就把我们的数据库搞垮，导致整个服务瘫痪!

解决方法:

解决方法：在缓存的时候给过期时间加上一个随机值，这样就会大幅度的减少缓存在同一时间过期。

对于“Redis挂掉了，请求全部走数据库”这种情况，我们可以有以下的思路:

事发前：实现Redis的高可用(主从架构+Sentinel 或者Redis Cluster)，尽量避免Redis挂掉这种情况发生。

事发中：万一Redis真的挂了，我们可以设置本地缓存(ehcache)+限流(hystrix)，尽量避免我们的数据库被干掉(起码能保证我们的服务还是能正常工作的)

事发后：redis持久化，重启后自动从磁盘上加载数据，快速恢复缓存数据。

96. 缓存穿透

什么是缓存穿透
缓存穿透是指查询一个一定不存在的数据。由于缓存不命中，并且出于容错考虑，如果从数据库查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，失去了缓存的意义
请求的数据在缓存大量不命中，导致请求走数据库。
缓存穿透如果发生了，也可能把我们的数据库搞垮，导致整个服务瘫痪！

如何解决缓存穿透
由于请求的参数是不合法的(每次都请求不存在的参数)，于是我们可以使用布隆过滤器(BloomFilter)或者压缩filter提前拦截，不合法就不让这个请求到数据库层！

当我们从数据库找不到的时候，我们也将这个空对象设置到缓存里边去。下次再请求的时候，就可以从缓存里边获取了。

这种情况我们一般会将空对象设置一个较短的过期时间。

97. 缓存击穿

某一个热点key，在不停地扛着高并发，当这个热点key在失效的一瞬间，持续的高并发访问就击破缓存直接访问数据库，导致数据库宕机。

设置热点数据"永不过期" 加上互斥锁：上面的现象是多个线程同时去查询数据库的这条数据，那么我们可以在第一个查询数据的请求上使用一个互斥锁来锁住它 其他的线程走到这一步拿不到锁就等着，等第一个线程查询到了数据，然后将数据放到redis缓存起来。
后面的线程进来发现已经有缓存了，就直接走缓存

总结：
雪崩是大面积的key缓存失效；穿透是redis里不存在这个缓存key；击穿是redis某一个热点key突然失效，最终的受害者都是数据库。

98. redis为什么这么快

1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是O(1)；

2、数据结构简单，对数据操作也简单，Redis 中的数据结构是专门进行设计的；

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4、使用多路 I/O 复用模型，非阻塞 IO；

5、使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求

99. memcache有哪些应用场景

作为数据库的前端缓存应用： 完整缓存，静态缓存。
比如 商品分类，商品信息

100. memcache 服务特点及工作原理

a. 完全基于内存缓存的
b、节点之间相互独立
c、C/S模式架构，C语言编写，总共2000行代码。
d、异步Ⅰ/O 模型，使用libevent作为事件通知机制。
e、被缓存的数据以key/value键值对形式存在的。
f、全部数据存放于内存中，无持久性存储的设计，重启服务器，内存里的数据会丢失。
g、当内存中缓存的数据容量达到启动时设定的内存值时，就自动使用LRU算法删除过期的缓存数据。
h、可以对存储的数据设置过期时间，这样过期后的数据自动被清除，服务本身不会监控过期，而是在访问的时候查看key的时间戳,判断是否过期。
j、memcache会对设定的内存进行分块，再把块分组，然后再提供服务

101. memcached是如何做身份验证的?

没有身份认证机制，如果你想限制访问，可以使用防火墙

102. mongoDB是什么?

MongoDB是一个文档数据库，提供好的性能，领先的非关系型数据库。采用BSON存储文档数据。
BSON（）是一种类json的一种二进制形式的存储格式，简称Binary JSON.
相对于json多了date类型和二进制数组。

103. mongodb的优势

面向文档的存储：以 JSON 格式的文档保存数据。
任何属性都可以建立索引。
复制以及高可扩展性。
自动分片。
丰富的查询功能。
快速的即时更新。

104. mongodb使用场景

大数据
内容管理系统
移动端Apps
数据管理

105. kafka 中的ISR，AR代表什么，ISR伸缩又代表什么

ISR:In-Sync Replicas 副本同步队列
AR:Assigned Replicas 所有副本
ISR是由leader维护，follower从leader同步数据有一些延迟（包括延迟时间replica.lag.time.max.ms和延迟条数replica.lag.max.messages两个维度, 当前最新的版本0.10.x中只支持replica.lag.time.max.ms这个维度），任意一个超过阈值都会把follower剔除出ISR, 存入OSR（Outof-Sync Replicas）列表，新加入的follower也会先存放在OSR中。AR=ISR+OSR。

106.kafka中的broker 是干什么的

broker 是消息的代理，Producers往Brokers里面的指定Topic中写消息，Consumers从Brokers里面拉取指定Topic的消息，然后进行业务处理，broker在中间起到一个代理保存消息的中转站。

107. kafka中的 zookeeper 起到什么作用，可以不用zookeeper么

zookeeper 是一个分布式的协调组件，早期版本的kafka用zk做meta信息存储，consumer的消费状态，group的管理以及 offset的值。考虑到zk本身的一些因素以及整个架构较大概率存在单点问题，新版本中逐渐弱化了zookeeper的作用。新的consumer使用了kafka内部的group coordination协议，也减少了对zookeeper的依赖，

但是broker依然依赖于ZK，zookeeper 在kafka中还用来选举controller 和 检测broker是否存活等等

108. kafka follower如何与leader同步数据

Kafka的复制机制既不是完全的同步复制，也不是单纯的异步复制。完全同步复制要求All Alive Follower都复制完，这条消息才会被认为commit，这种复制方式极大的影响了吞吐率。

而异步复制方式下，Follower异步的从Leader复制数据，数据只要被Leader写入log就被认为已经commit，这种情况下，如果leader挂掉，会丢失数据，kafka使用ISR的方式很好的均衡了确保数据不丢失以及吞吐率。Follower可以批量的从Leader复制数据，而且Leader充分利用磁盘顺序读以及send file(zero copy)机制，这样极大的提高复制性能，内部批量写磁盘，大幅减少了Follower与Leader的消息量差。

109. kafka 为什么那么快

- Cache Filesystem Cache PageCache缓存
- 顺序写 由于现代的操作系统提供了预读和写技术，磁盘的顺序写大多数情况下比随机写内存还要快。
- Zero-copy 零拷技术减少拷贝次数
- Batching of Messages 批量量处理。合并小的请求，然后以流的方式进行交互，直顶网络上限。
- Pull 拉模式 使用拉模式进行消息的获取消费，与消费端处理能力相符。

110. Kafka中的消息是否会丢失和重复消费?

要确定Kafka的消息是否丢失或重复，从两个方面分析入手：消息发送和消息消费。

1、消息发送

```
Kafka消息发送有两种方式：同步（sync）和异步（async），默认是同步方式，可通过producer.type属性进行配置。Kafka通过配置request.required.acks属性来确认消息的生产：
```

0---表示不进行消息接收是否成功的确认；
1---表示当Leader接收成功时确认；
-1---表示Leader和Follower都接收成功时确认；
综上所述，有6种消息生产的情况，下面分情况来分析消息丢失的场景：

（1）acks=0，不和Kafka集群进行消息接收确认，则当网络异常、缓冲区满了等情况时，消息可能丢失；

（2）acks=1、同步模式下，只有Leader确认接收成功后但挂掉了，副本没有同步，数据可能丢失；

2、消息消费

Kafka消息消费有两个consumer接口，Low-level API和High-level API：

Low-level API：消费者自己维护offset等值，可以实现对Kafka的完全控制；

High-level API：封装了对parition和offset的管理，使用简单；

如果使用高级接口High-level API，可能存在一个问题就是当消息消费者从集群中把消息取出来、并提交了新的消息offset值后，还没来得及消费就挂掉了，那么下次再消费时之前没消费成功的消息就"诡异"的消失了；

解决办法：

> 针对消息丢失：同步模式下，确认机制设置为-1，即让消息写入Leader和Follower之后再确认消息发送成功；异步模式下，为防止缓冲区满，可以在配置文件设置不限制阻塞超时时间，当缓冲区满时让生产者一直处于阻塞状态；
>
> 针对消息重复：将消息的唯一标识保存到外部介质中，每次消费时判断是否处理过即可。

111. 为什么Kafka不支持读写分离?

在 Kafka 中，生产者写入消息、消费者读取消息的操作都是与 leader 副本进行交互的，从 而实现的是一种主写主读的生产消费模型
Kafka 并不支持主写从读，因为主写从读有 2 个很明 显的缺点:

(1)数据一致性问题。数据从主节点转到从节点必然会有一个延时的时间窗口，这个时间 窗口会导致主从节点之间的数据不一致。某一时刻，在主节点和从节点中 A 数据的值都为 X， 之后将主节点中 A 的值修改为 Y，那么在这个变更通知到从节点之前，应用读取从节点中的 A 数据的值并不为最新的 Y，由此便产生了数据不一致的问题。

(2)延时问题。类似 Redis 这种组件，数据从写入主节点到同步至从节点中的过程需要经 历网络→主节点内存→网络→从节点内存这几个阶段，整个过程会耗费一定的时间。而在 Kafka 中，主从同步会比 Redis 更加耗时，它需要经历网络→主节点内存→主节点磁盘→网络→从节 点内存→从节点磁盘这几个阶段。对延时敏感的应用而言，主写从读的功能并不太适用。

112. 什么是消费者组?

消费者组是 Kafka 独有的概念，如果面试官问这 个，就说明他对此是有一定了解的。我先给出标准答案:
1、定义：即消费者组是 Kafka 提供的可扩展且具有容错性的消费者机制。
2、原理：在 Kafka 中，消费者组是一个由多个消费者实例 构成的组。多个实例共同订阅若干个主题，实现共同消费。同一个组下的每个实例都配置有 相同的组 ID，被分配不同的订阅分区。当某个实例挂掉的时候，其他实例会自动地承担起 它负责消费的分区。

此时，又有一个小技巧给到你:消费者组的题目，能够帮你在某种程度上掌控下面的面试方向。

如果你擅长位移值原理，就不妨再提一下消费者组的位移提交机制;
如果你擅长 Kafka Broker，可以提一下消费者组与 Broker 之间的交互;
如果你擅长与消费者组完全不相关的 Producer，那么就可以这么说:"消费者组要消 费的数据完全来自于 Producer 端生产的消息，我对 Producer 还是比较熟悉的。"

113. Kafka 中的术语

- 代理（borker）: 一个kafka进程（kafka进程又被称为实例），被称为一个代理broker节点。
- 生产者（producer）

Producer将消息记录发送到Kafka集群指定的主题（Topic）中进行存储，同时生产者（Producer）也能通过自定义算法决定将消息记录发送到哪个分区（Partition）。
例如，通过获取消息记录主键（Key）的哈希值，然后使用该值对分区数取模运算，得到分区索引。

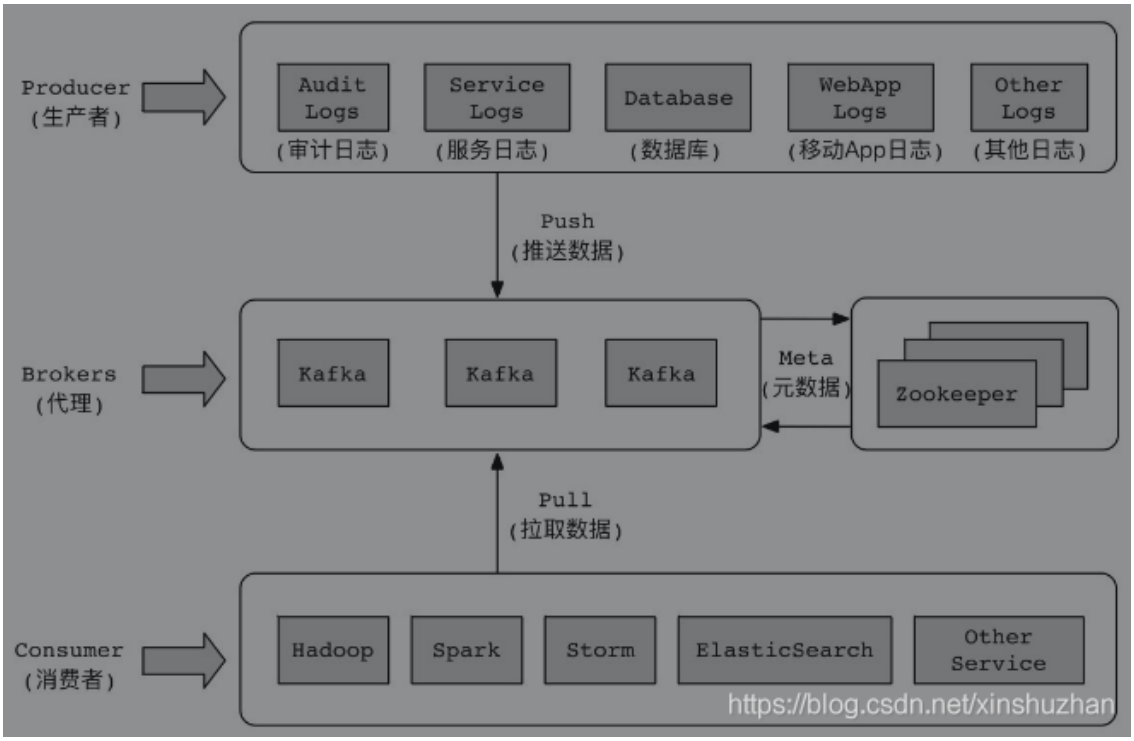
- 消费者Consumer
  消费者（Consumer）从Kafka集群指定的主题（Topic）中读取消息记录。在读取主题数据时，需要设置消费组名（GroupId）。如果不设置，则Kafka消费者会默认生成一个消费组名称。
- 消费者组： Consumer Group
  消费者程序在读取Kafka系统主题（Topic）中的数据时，通常会使用多个线程来执行。一个消费者组可以包含一个或多个消费者程序，使用多分区和多线程模式可以极大提高读取数据的效率。

一般而言，一个消费者对应一个线程

- 主题Topic
  Kafka系统通过主题来区分不同业务类型的消息记录。例如，用户登录数据存储在主题A中，用户充值记录存储在主题B中，则如果应用程序只订阅了主题A，而没有订阅主题B，那该应用程序只能读取主题A中的数据
- 分区（Partition）
  每一个主题（Topic）中可以有一个或者多个分区（Partition）。在Kafka系统的设计思想中，分区是基于物理层面上的，不同的分区对应着不同的数据文件。Kafka通过分区（Partition）来支持物理层面上的并发读写，以提高Kafka集群的吞吐量。

一个分区只对应一个代理节点（Broker），一个代理节点可以管理多个分区。

- 副本（replication）
  在Kafka系统中，每个主题（Topic）在创建时会要求指定它的副本数，默认是1。通过副本（Replication）机制来保证Kafka分布式集群数据的高可用性
  在创建主题时，主题的副本系数值应如下设置：（1）若集群数量大于等于3，则主题的副本系数值可以设置为3；（2）若集群数量小于3，则主题的副本系数值可以设置为小于等于集群数量值。例如，集群数为2，则副本系数可以设置为1或者2；集群数为1，则副本系数只能设置为1。
- 记录（Record）
  被实际写入到Kafka集群并且可以被消费者应用程序读取的数据，被称为记录（Record）。每条记录包含一个键（Key）、值（Value）和时间戳（Timestamp）。
- replica：
  partition 的副本，保障 partition 的高可用。
- leader：
  replica 中的一个角色， producer 和 consumer 只跟 leader 交互。
- follower：
  replica 中的一个角色，从 leader 中复制数据。
- controller：
  kafka 集群中的其中一个服务器，用来进行 leader election 以及 各种 failover。
- zookeeper：
  kafka 通过 zookeeper 来存储集群的 meta 信息。



生产者（Producer）负责写入消息数据。将审计日志、服务日志、数据库、移动App日志，以及其他类型的日志主动推送到Kafka集群进行存储。·

消费者（Consumer）负责读取消息数据。例如，通过Hadoop的应用接口、Spark的应用接口、Storm的应用接口、ElasticSearch的应用接口，以及其他自定义服务的应用接口，主动拉取Kafka集群中的消息数据。

114. kafka适用于哪些场景

1. 日志收集
2. 消息系统
3. 用户轨迹（记录浏览器用户或者app用户产生的各种记录，点击和搜索浏览等）
4. 记录运营监控数据
5. 实现流处理

115. Kafka写入流程：

1.producer 先从 zookeeper 的 "/brokers/.../state" 节点找到该 partition 的 leader

2. producer 将消息发送给该 leader
3. leader 将消息写入本地 log
4. followers 从 leader pull 消息，写入本地 log 后 leader 发送 ACK
5. leader 收到所有 ISR 中的 replica 的 ACK 后，增加 HW（high watermark，最后 commit 的 offset） 并向 producer 发送 ACK

116. zabbix有哪些组件

1）Zabbix Server：负责接收agent发送的报告信息的核心组件，所有配置、统计数据及操作数据均由其组织进行

2）Database Storage：专用于存储所有配置信息，以及有zabbix收集的数据

3）Web interface（frontend）：zabbix的GUI接口，通常与server运行在同一台机器上

4）Proxy：可选组件，常用于分布式监控环境中，代理Server收集部分被监控数据并统一发往Server端

5）Agent：部署在被监控主机上，负责收集本地数据并发往Server端或者Proxy端

117. zabbix的两种监控模式

Zabbix agent检测分为两种模式：主动模式和被动模式
被动模式，也是默认的Zabbix监控模式，被动模式是相对于proxy来说的。proxy主动发送数据就是主动模式，proxy等待server的请求再发送数据就是被动模式。

使用zabbix主动模式的好处：可以监控不可达的远程设备；监控本地网络不稳定区域；当监控项目数以万计的时候使用代理可以有效分担zabbix server的压力；简化zabbix分布式监控的维护。

被动模式：由server向agent发出指令获取数据， 即agent被动的去获取数据并返回给server，server周期性的向agent 索取数据， 这总模式的最大问题就是会加大server的工作量， 在数百台服务器的环境下server不能及时获取到最新数据， 但这也是默认的工作方式。

118. 一个监控系统的运行流程

zabbix agent 需要安装到被监控的主机上，它负责定期收集各项数据，并发送到 zabbix server 端，zabbix server 将数据存储到数据库中，zabbix web 根据数据在前端进行展示和绘图。

119. zabbix的工作进程

默认情况下 zabbix 包含6个进程：zabbix_agentd、zabbix_get、zabbix_proxy、zabbix_sender、zabbix_server，另外一个zabbix_java_gateway是可选的，这个需要单独安装。

- zabbix_get

zabbix 工具，单独使用的命令，通常在 server 或者 proxy 端执行获取远程客户端信息的命令。通常用于排错。例如在 server 端获取不到客户端的内存数据，可以使用 zabbix_get 获取客户端的内容的方式来做故障排查。

- zabbix_sender
  zabbix 工具，用于发送数据给 server 或者 proxy，通常用于耗时比较长的检查。很多检查非常耗时间，导致 zabbix 超时。于是在脚本执行完毕之后，使用 sender 主动提交数据。
- zabbix_server

zabbix 服务端守护进程。zabbix_agentd、zabbix_get、zabbix_sender、zabbix_proxy、zabbix_java_gateway的数据最终都是提交到server（说明：当然不是数据都是主动提交给zabbix_server，也有的是 server 主动去取数据，主要看是使用了什么模式进行工作）

- zabbix_proxy
  zabbix 代理守护进程。功能类似server，唯一不同的是它只是一个中转站，它需要把收集到的数据提交/被提交到 server 里。
- zabbix_agentd
  客户端守护进程，此进程收集客户端数据，例如cpu负载、内存、硬盘使用情况等。

120. zabbix常用术语

1. 主机（host）：要监控的网络设备，可由IP或DNS名称指定；
2. 主机组（hostgroup）：主机的逻辑容器，可以包含主机和模板，但同一个组织内的主机和模板不能互相链接；主机组通常在给用户或用户组指派监控权限时使用；

3. 监控项（item）：一个特定监控指标的相关的数据；这些数据来自于被监控对象；item是zabbix进行数据收集的核心，相对某个监控对象，每个item都由"key"标识；
4. 触发器（trigger）：一个表达式，用于评估某监控对象的特定item内接收到的数据是否在合理范围内，也就是阈值；接收的数据量大于阈值时，触发器状态将从"OK"转变为"Problem"，当数据再次恢复到合理范围，又转变为"OK"；
5. 事件（event）：触发一个值得关注的事情，比如触发器状态转变，新的agent或重新上线的agent的自动注册等；
6. 动作（action）：指对于特定事件事先定义的处理方法，如发送通知，何时执行操作；
7. 报警升级（escalation）：发送警报或者执行远程命令的自定义方案，如每隔5分钟发送一次警报，共发送5次等；
8. 媒介（media）：发送通知的手段或者通道，如Email、Jabber或者SMS等；
9. 通知（notification）：通过选定的媒介向用户发送的有关某事件的信息； 远程命令（remote command）：预定义的命令，可在被监控主机处于某特定条件下时自动执行；
10. 模板（template）：用于快速定义被监控主机的预设条目集合，通常包含了item、trigger、graph、screen、application以及low-level
11. discovery rule；模板可以直接链接至某个主机； 应用（application）：一组item的集合；
12. web场景（webscennario）：用于检测web站点可用性的一个活多个HTTP请求； 前端（frontend）：Zabbix的web接口；

121. zabbix自定义发现是怎么做的?

zabbix发现有3种类型：

1. 自动网络发现
2. 主动客户端自动注册
3. 低级别的发现

实现方式： 规则+动作

1. 首先需要在模板当中创建一个自动发现的规则，这个地方只需要一个名称和一个键值。
   2、过滤器中间要添加你需要的用到的值宏。

3、然后要创建一个监控项原型，也是一个名称和一个键值。

4、然后需要去写一个这样的键值的收集。

自动发现实际上就是需要首先去获得需要监控的值，然后将这个值作为一个新的参数传递到另外一个收集数据的item里面去。

122. 微信报警

首先要有企业微信，然后需要几个参数，从企业微信后台获取：Agetid,Secret,成员账号，组织部门ID，应用ID
使用Python或者shell脚本来实现微信报警
就是一个获取企业微信接口，和调用邮件接口来发送信息的过程。

123. zabbix客户端如何批量安装

可以使用ansible+shell脚本自动化部署 zabbix-agent软件包和管理配置文件。

124. zabbix分布式是如何做的

使用zabbix proxy
zabbix proxy 可以代替 zabbix server 收集性能和可用性数据,然后把数据汇报给 zabbix server,并且在一定程度上分担了zabbix server 的压力.

此外，当所有agents和proxies报告给一个Zabbix server并且所有数据都集中收集时，使用proxy是实现集中式和分布式监控的最简单方法。

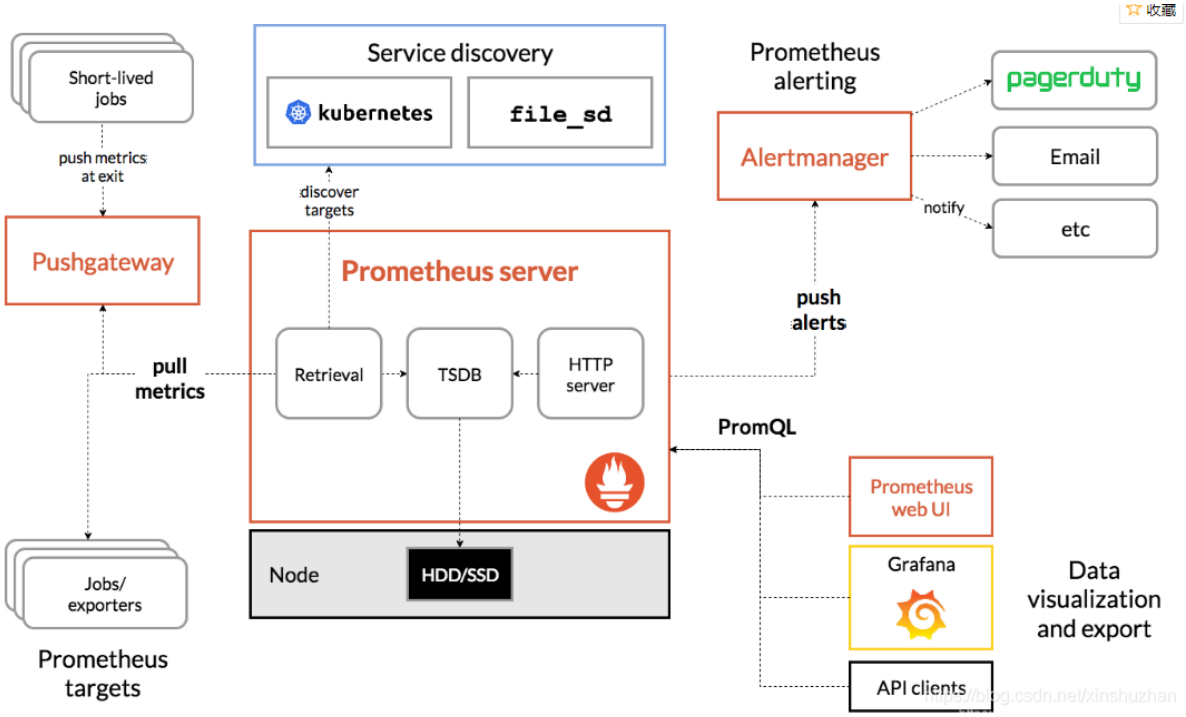
zabbix proxy 仅仅需要一条 tcp 连接到 zabbix server,所以防火墙上仅仅需要加上一条规则即可。

proxy 收集到数据之后，首先将数据缓存在本地,然后在一定得时间之后传递给 zabbix server，这样就不会因为服务器的任何临时通信问题而丢失数据。这个时间由 proxy配置文件中参数 ProxyLocalBuffer 和 ProxyOfflineBuffer 决定。

125. zabbix proxy 的使用场景

1.远程监控设备
2.监控网络不稳定的区域
3.当zabbix监控大型架构的时候
4.为了分布式

126. prometheus工作原理



- prometheus server定期从配置好的jobs或者exporters中拉取metrics，或者接收来自Pushgateway发送过来的metrics，或者从其它的Prometheus server中拉metrics。
- Prometheus server在本地存储收集到的metrics，并运行定义好的alerts.rules，记录新的时间序列或者向Alert manager推送警报。
- Alertmanager根据配置文件，对接收到的警报进行处理，发出告警。
- 在图形界面中，可视化采集数据。

127. prometheus组件

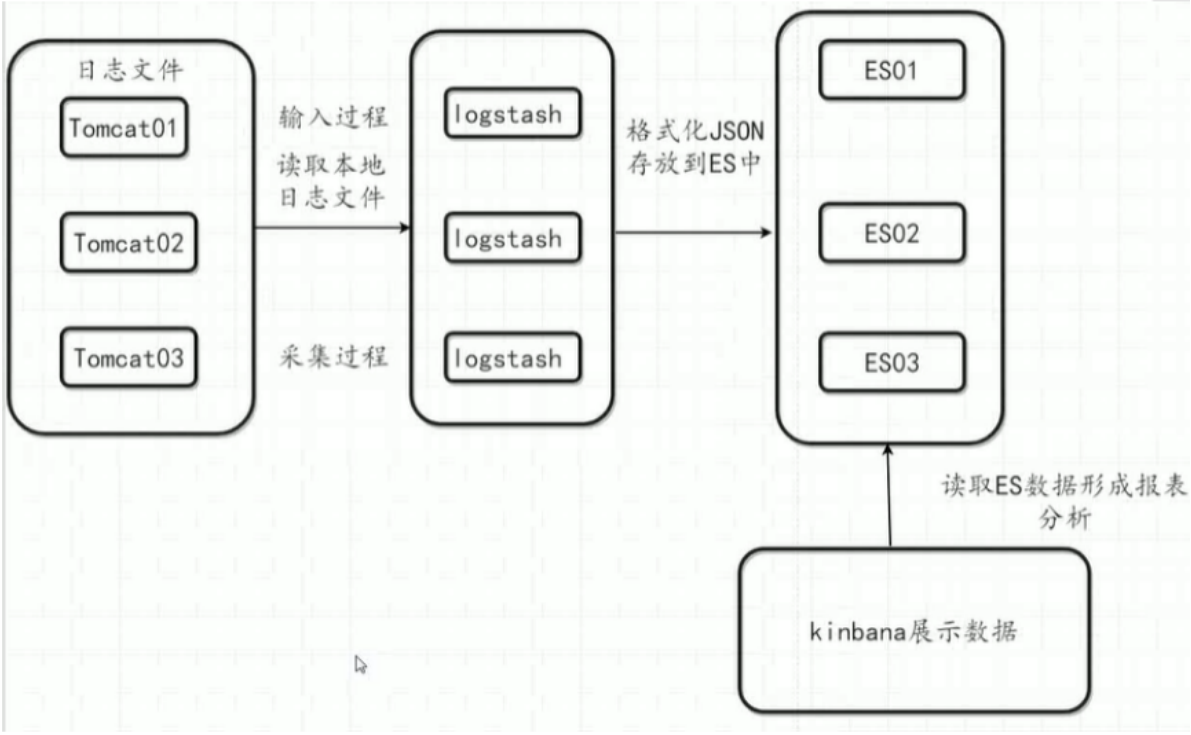
1.Prometheus Server：
Prometheus Sever是Prometheus组件中的核心部分，负责实现对监控数据的获取，存储及查询。Prometheus Server可以通过静态配置管理监控目标，也可以配合使用Service Discovery的方式动态管理监控目标，并从这些监控目标中获取数据。其次Prometheus Sever需要对采集到的数据进行存储，Prometheus Server本身就是一个实时数据库，将采集到的监控数据按照时间序列的方式存储在本地磁盘当中。Prometheus Server对外提供了自定义的PromQL，实现对数据的查询以及分析。另外Prometheus Server的联邦集群能力可以使其从其他的Prometheus Server实例中获取数据。

2 Exporters：
Exporter将监控数据采集的端点通过HTTP服务的形式暴露给Prometheus Server，Prometheus Server通过访问该Exporter提供的Endpoint端点，即可以获取到需要采集的监控数据。可以将Exporter分为2类：
直接采集：这一类Exporter直接内置了对Prometheus监控的支持，比如cAdvisor，Kubernetes，Etcd，Gokit等，都直接内置了用于向Prometheus暴露监控数据的端点。
间接采集：原有监控目标并不直接支持Prometheus，因此需要通过Prometheus提供的Client Library编写该监控目标的监控采集程序。例如：Mysql Exporter，JMX Exporter，Consul Exporter等。

3 AlertManager：
在Prometheus Server中支持基于Prom QL创建告警规则，如果满足Prom QL定义的规则，则会产生一条告警。在AlertManager从 Prometheus server 端接收到 alerts后，会进行去除重复数据，分组，并路由到对收的接受方式，发出报警。常见的接收方式有：电子邮件，pagerduty，webhook 等。

4 PushGateway:
Prometheus数据采集基于Prometheus Server从Exporter pull数据，因此当网络环境不允许Prometheus Server和Exporter进行通信时，可以使用PushGateway来进行中转。通过PushGateway将内部网络的监控数据主动Push到Gateway中，Prometheus Server采用针对Exporter同样的方式，将监控数据从PushGateway pull到Prometheus Server。

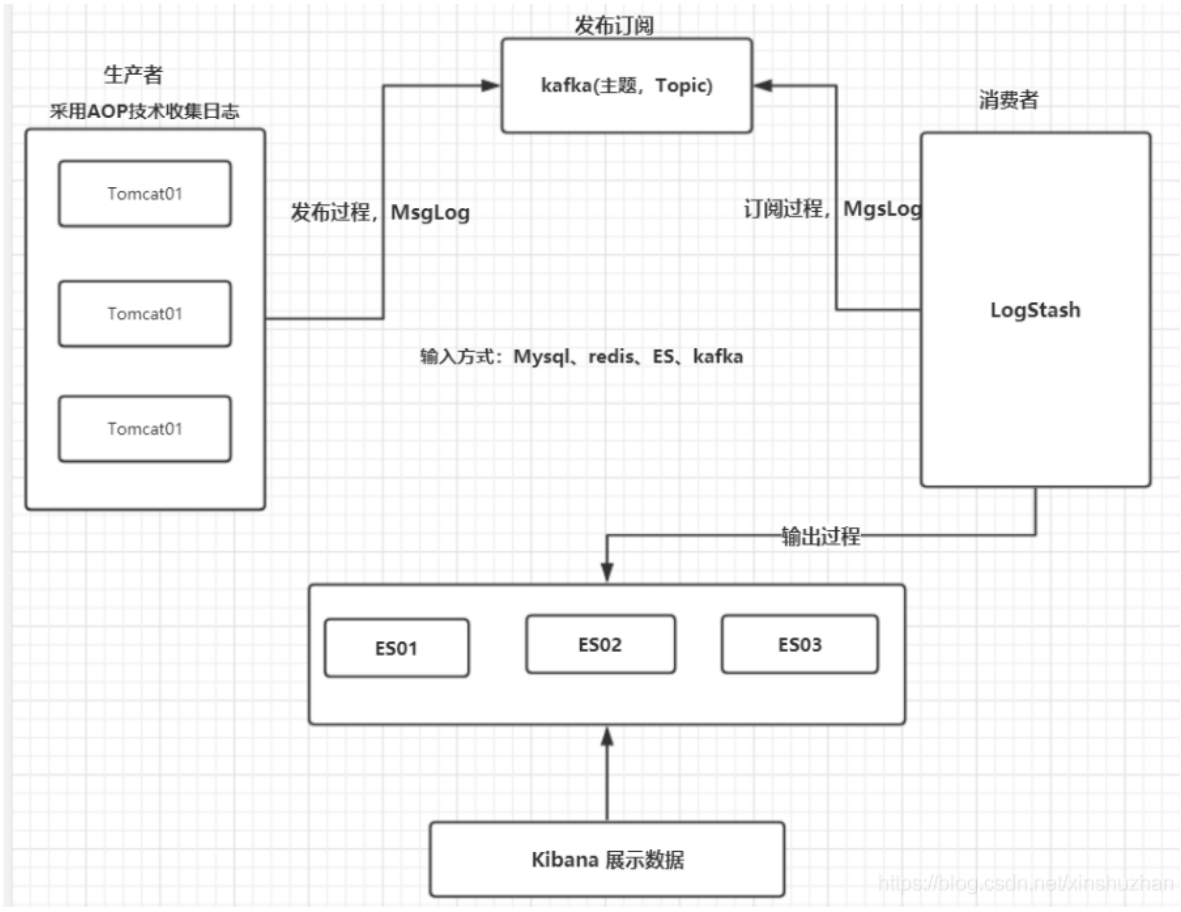
128. ELK工作流程



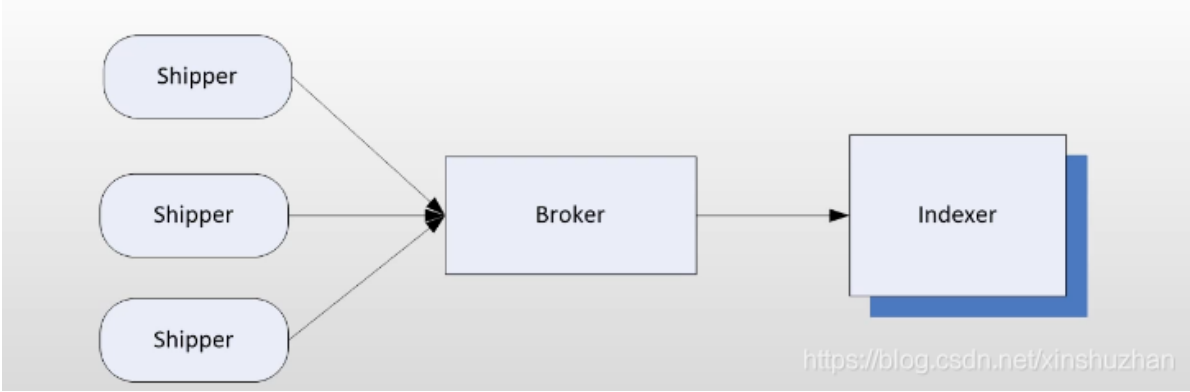
129. logstash的输入源有哪些?

Logstash输入来源有那些?
本地文件、kafka、数据库、mongdb、redis。

ELK+Kafka 流程图



130. logstash的架构



input: 数据收集
filter： 数据加工
output: 数据输出

shipper: 用于收集日志数据，监控本地日志数据的变化，及时把日志文件的最新内容收集起来，然后经过加工，过滤，输出到broker

broker: 相当于日志的hub,用来连接多个shipper和多个indexer

indexer： 从boker读取文本，经过加工，过滤输出到指定介质。 redis服务器就是官方推荐的broker

131. ELK相关的概念

Node： 装有一个 ES 服务器的节点。
Cluster： 有多个Node组成的集群
Document： 一个可被搜素的基础信息单元
Index： 拥有相似特征的文档的集合

Type： 一个索引中可以定义一种或多种类型
Filed： 是 ES 的最小单位，相当于数据的某一列
Shards： 索引的分片，每一个分片就是一个 Shard
Replicas： 索引的拷贝

132. es常用的插件

1. 公司的代码用的gitlab,所以我们使用gitlab插件来拉取代码
2. gitlab webhook 支持gogs 代码仓库的触发
3. 在构建过程中，需要用到脚本语言，比如groovy,powershell，这些脚本语言就需要相应的插件来执行脚本
4. 构建完了之后需要发送邮件，可以使用 Email Extension Plugin来定义发送邮件
5. jenkins与钉钉结合实现告警： DingDing plugin
6. 能够支持一件构建的: bulid pipeline plugin ，一次构建几十个jobs
7. 传送文件到目标服务然后执行命令： publish over ssh
8. git插件，用于托管自动化测试代码和协作开发
9. build history metrics plugin 用于job进行统计的插件
   10.jenkins与ansible结合： ansibleplaybook

134. zabbix你都监控哪些参数

说到监控，在运维这个行业其实有很多开源的监控方案，目前最常见的就是zabbix+grafana， 我工作那时候还是用cacti和nagios的比较多。

还记得以前去面试，面试官来了一句，zabbix会搭建吗，会的话你在这搭建下，30分钟搭建出来就入职。

不管是zabbix，还是其他的开源监控，说到底都是在做五件事：

1. 数据的采集
2. 采集过来的数据存储
3. 把存储起来的数据进行分析
4. 把分析的结果使用图标展示
5. 把有问题的地方采用各种方式告警。

而我们要监控的也无非是5大块，服务器，中间件，数据库，网络设备，应用。

监控指标举例

1. 监控web服务

- web服务是否正常
- 业务（网页是否能访问、是否可以完成下订单、注册用户）
- 服务的响应时间
- 服务的并发量（活动用户、非活动用户）

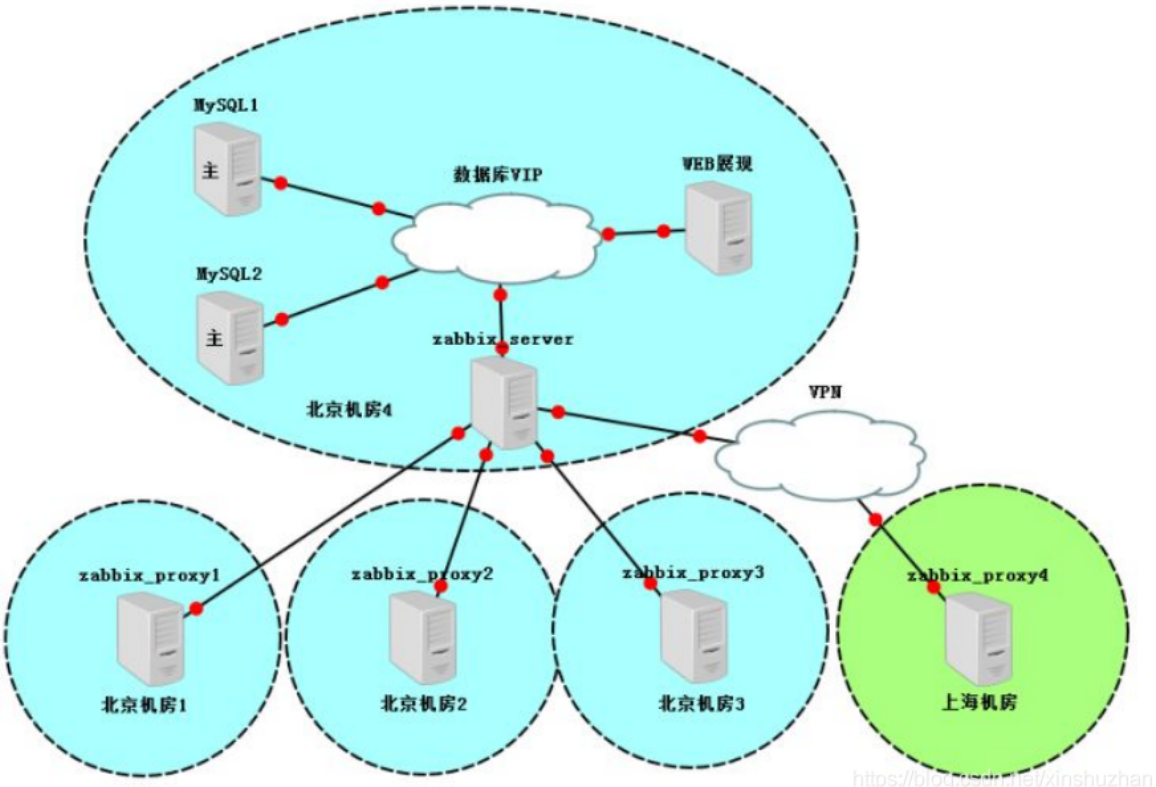
2. 监控数据库

- 监控磁盘使用情况
- 监控内存内存使用
- 查看并发连接数量
- 检查数据库执行增删改查的频率
- 检查主从状态
- 检查数据库的备份情况

3. 服务器监控

- 磁盘
  - 使用率
  - inode数
  - block数
  - 读写速率
- CPU
  - 监控cpu负载
  - 监控使用cpu资源最多的进程
- 内存
  - 使用率
  - 缓冲区
  - 缓存区
  - 交换分区大小
- 网络
  - 监控每个网卡的上先行速率
  - 监控占用网络带宽见多的进程
  - 监控数据包的丢包
  - 监控网络数据包的阻塞情况
- 进程
  - 当前系统中的总进程数
  - 监控特定的程序的进程数

企业监控架构图



采集服务器： 1台， 8C8G 100G硬盘
数据库服务器：2台 8核16G 3个IP地址
web服务器 1台 4核心4G
采集代理服务器: n台， 8核8G 100G硬盘， 根据主机和网段增加

如何回答问题

示例：
我们主要用的是zabbix作为监控，主要监控 数据库，web，网络，中间件和一些其他应用，
比如我们监控mysql，主要关注磁盘，内存，并发数，数据库的增啥改查频率，主从状态等等。

数据库性能方面，主要关注：

- 查询吞吐量
- 查询执行性能
- 连接情况
- 缓冲池使用情况

补充扩展

zabbix常用的术语

```
主机（host）：              要监控的网络设备，可由ip或DNS名称指定。
主机组（host group）：      主机的逻辑容器，可以包含主机和模板，但同一个组内的主机和模版不能互相链接，主机组通常在组用户或用户组指派监控权限时使用。
监控项（item）：            一个特定监控指标的相关的数据，这些数据来自于被监控对象，item是zabbix进行数据收集的核心，没有item，就没有数据，每个item都由“key”进行标识。
触发器（trigger）：         一个表达式，用于评估某监控对象的某特定item内所接收到的数据是否在合理范围内，即阈值；接收到的数据量大于阈值时，触发器状态将从“OK”转变为“Problem”，当数据量再次回到合理范围时，其状态将从“Problem”转换回“OK”。
事件（event）：             即发生的一个值得关注的事件，如触发器的状态转变，新的agent或重新上线的agent的自动注册等。
动作（action）：            指对于特定事件事先定义的处理方法，通过包含操作（如发送通知）和条件（何时执行操作）。
报警升级（escalation）：    发送警报或执行远程命令的自定方案，如每隔多长时间发送一次警报，共发送多少次。
媒介（media）：             发送通知的手段或通道，如Email，Jabber或SMS等。
通知（motification）：      通过选定的媒介向用户发送的有关某事件的信息。
远程命令（remote command）：预定义的命令，可在被监控主机处于某特定条件下时自动执行。
模版（template）：          用于快速定义被监控主机的预设条目集合，通常包含了item，trigger，graph，screen，application，low-level discovery rule，模板可以直接链接到单个主机。
应用（application）：       用于检测web站点可用性的一个或多个HTTP请求。
前端（frontend）：          zabbix的web接口。
```

单纯对于面试来说这个题目还是很好回答的，但真正操作起来会稍微麻烦点，因为节点数多，监控的项多，好在公司里都有现成的。

当然面试官问完这个问题会可能会问的问题有：

1. zabbix监控你都用过哪些模板
2. 你都写过哪些模板
3. 微信报警怎么做的
4. 如何添加一台主机

这些问题也需要提前准备。
切记： 面试只分为两种，准备过和没有准备过。 我们不打无准备之仗。

后记：
1.硬件监控——路由器、交换机、防火墙

2.系统监控——cpu、内存、磁盘、网络、进程、tcp

3.服务监控——nginx、php、tomcat、redis、memcache、mysql

4.web监控——响应时间、加载时间、渲染时间，页面是不是200

5.日志监控——ELK、（收集、存储、分析、展示）日志

6.安全监控——firewalld、WAF(nginx+lua)、安全宝、牛盾云、安全狗

135. MySQL同步和半同步

一般面试官问这个问题，都伴随着另外一个问题，叫：小伙子，你说一下MySQL主从原理?
当你回答完原理之后，就会问到关于MySQL主从的几种方式的问题。

一般我们的主从同步，分为：半同步复制和异步复制：
MySQL主从复制默认采用的是异步复制的机制进行同步，所谓的异步指的是主库执行完客户端提交的事务之后立即将结果返回给客户端，并不关心从库是否已经接收道binlog并同步。

如果从库还没有接收到binlog的时候主库发生宕机，那么进行主从切换后的从库数据并不能和主库完全一致。为了解决这个问题，从MySQL 5.5开始以插件的形式支持半同步复制（另需要注意由于半同步复制仅与默认复制通道（for channel “”）兼容，不支持与多源复制混用。）。半同步复制中，当一个客户端提交事务时主库执行完事务并不会立即响应该客户端，而是等待至少有一个从库接收到了binlog并将事务写到自己的relay log中，最后收到从库的ACK响应后才返回给客户端。

半同步这边根据存储引擎处理的逻辑不同又分为after-commit和after-sync两种：

after-commit
主库未收到从库的ACK消息之前，虽然不会让发起事务的会话收到事务提交结果，但是会在存储引擎层先执行提交。这样一来虽然提交事务的会话无法知道事务结果，但其他会话在主库上却是可以正常查询的。如果发生主从切换，那么原本在主库可以查询到的数据，在从库却丢失了

after-sync
也可以叫做增强半同步复制或者无损复制，是MySQL 5.7默认的半同步方式。主库未收到从库的ACK消息之前，发起事务的会话收无法到事务提交结果，存储引擎层也不会执行提交，解决了after-commit的缺陷，保证了数据的严格一致。由于OLTP应用场景大多数是读多写少，读不会产生binlog，所以即便开启无损复制，相比异步复制所损失的性能也并不多

半同步复制发生超时，可以由rel_semi_sync_master_timeout参数设置关闭半同步，转而使用异步。

136. CI/CD

这个问题在面试中也经常被问到，主要考察几个方面：

1. 你对新技术的了解?
2. 你们公司是如何落地的，来我们公司是否可以借鉴?

三个概念：
持续集成CI： 代码合并，构建，部署，测试都在一起，不断的执行这个过程，并对结果进行反馈。

通过这个过程，在未上线前去反复测试，减少上线后出现bug的几率

持续部署CD： 部署到测试环境，预生产环境，生产环境。

持续交付：CD： 把最终的产品发布到生产环境中，让用户去使用，在使用的过程中反馈结果。

这个过程涉及到运维，开发，测试。

CI/CD的最终目的是为了减少人工干预，实现自动化，提高产品交付的效率和质量。

CI/CD是一种解决方案，而实现这个方案又有很多种方法。

非容器化解决方案：
从开发上传代码到版本库中，jenkins拉取代码使用maven进行编译，然后部署上线。

这个过程很容易出问题，比如开发环境没问题，但是预生产环境出现问题，大家相互扯皮。

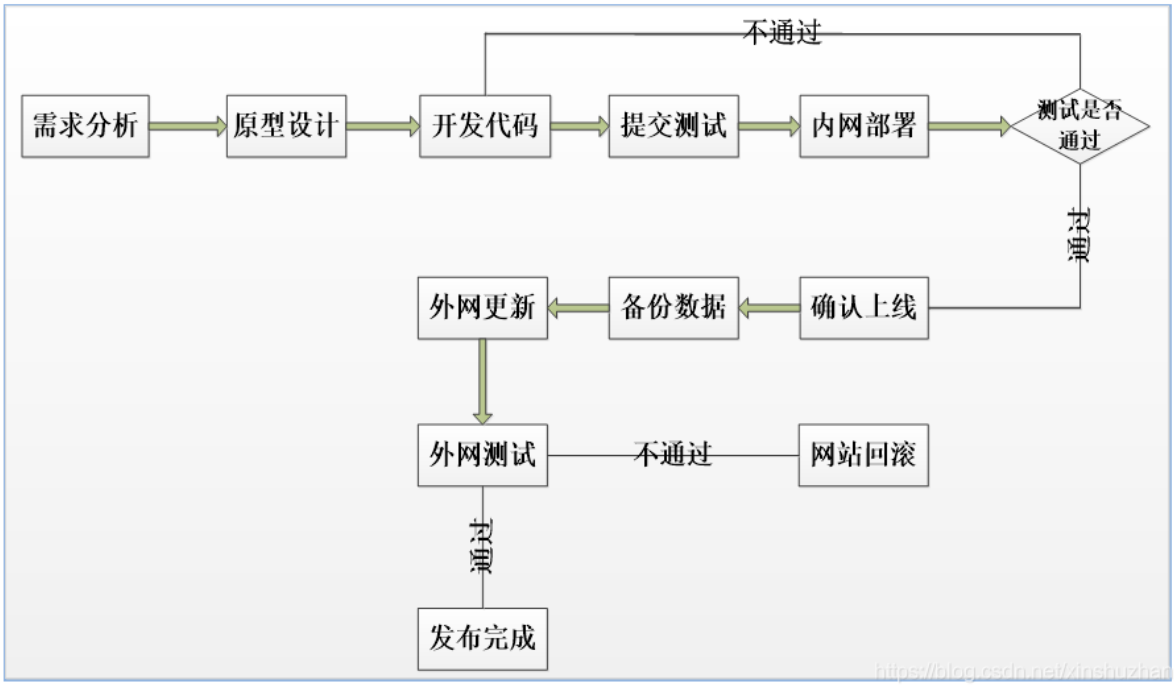
容器化解决方案

开发上传到代码到版本库，jenkins拉取进行编译，然后打包传到镜像仓库（harbor）.在harbor中进行测试。
如果不使用容器，脚本和配置文件会越来越多，越来越乱。

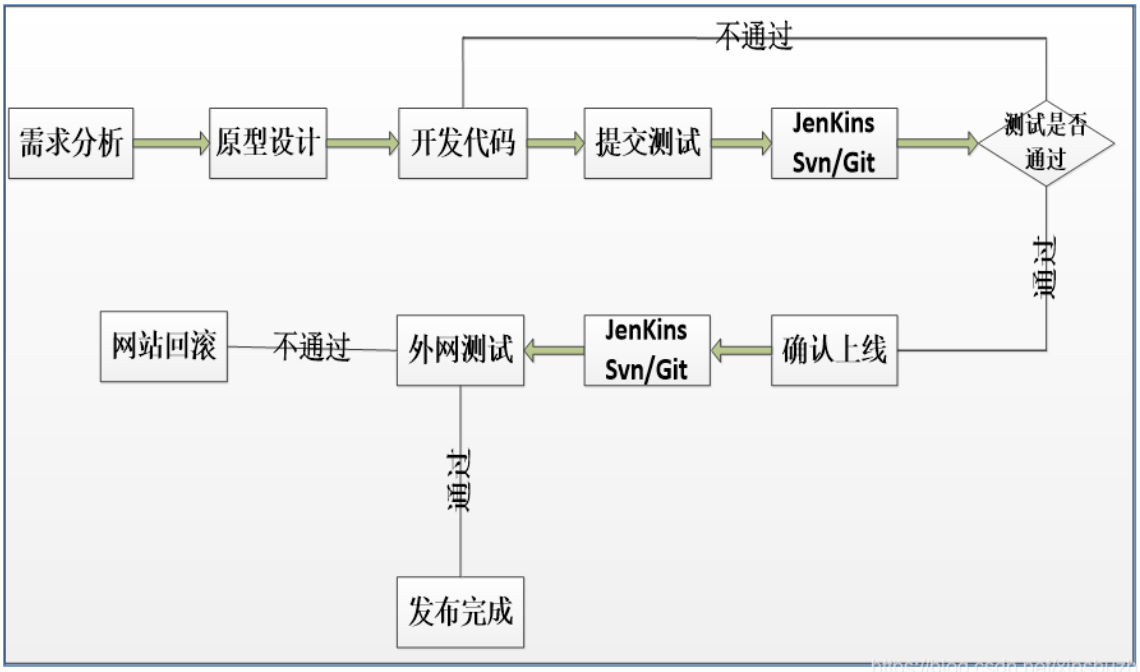
而使用了容器后，这个问题就容易解决了，在容器中，我们只需要交付镜像就可以了。

docker从镜像仓库中进行拉取，然后在进行部署。

补充：
传统网站部署流程：



使用jenkins后的部署流程



有了k8s之后：

开发写完代码上传到git版本仓库，jenkins拉取代码在maven进行编译，构建镜像，把镜像推送到镜像仓库，然后就可以在k8s中部署。 部署完成中，调用k8s的API进行健康检查（healthcheck）

使用k8s去做有什么好处?

1. 环境稳定
2. 服务不间断
3. 一次构建多环境去运行

K8S node管理

k8s中有三个组件与node交互，分别是node controller,kubelet,kubectl

在node的整个生命周期中，node controller充当多个角色，
第一个：在node注册入集群的时候，给他分配一个CIDR地址快。
第二个是维护node controller 内部节点可用列表，如果node controller检测node不健康，他就会向供应商询问节点虚拟机是否可用，如果不可应就从列表剔除。

第三个是监控节点的健康状态，当node controller检测node不可达时，负责将node状态中的not ready condition 跟新到 condition unknown， 然后将node上所有的pod删除，（默认不可达超过40S就会标记成condition unkunown,此时并没有发生删除pod的操作，系统会尝试重新联系node，如果恢复就被标记为node ready. 如果不可达超过5分钟，就会采取删除pod的操作）

每隔多少秒去检测一次节点的状态是： --node-monitor-period

node controller 源码中，monitor node health 是定时更新node的信息：

获取所有的node信息，按照哪些是新增的，哪些是需要删除的，以及哪些是需要重新规划的返回节点相应的信息
对新增node，删除node以及待规划的node做对应的处理
遍历所有node，更新node状态，调用tryupdatenodehealth方法
而kubelet运行在node上，维持运行中的pods以及k8s运行的环境，主要完成以下使命：

5. 监视分配给node的节点pods
6. 挂载pod所需要的volumes
7. 下载pod的secret
8. 通过docker来运行pod中的容器

9. 周期的执行pod中为容器定义的liveness探针
10. 上报pod的状态给系统的其他组件
11. 上报node的状态

k8s的监控指标

137 K8S监控指标

一般在公司里我们都是使用prometheus进行监控，先说一下prometheus的工作核心：
prometheus是使用 Pull （抓取）的方式去搜集被监控对象的 Metrics 数据（监控指标数据），然后，再把这些数据保存在一个 TSDB （时间序列数据库，比如 OpenTSDB、InfluxDB 等）当中，以便后续可以按照时间进行检索。

有了这套核心监控机制， Prometheus 剩下的组件就是用来配合这套机制的运行。比如 Pushgateway，可以允许被监控对象以 Push 的方式向 Prometheus 推送 Metrics 数据。

而 Alertmanager，则可以根据 Metrics 信息灵活地设置报警。当然， Prometheus 最受用户欢迎的功能，还是通过 Grafana 对外暴露出的、可以灵活配置的监控数据可视化界面。

kubernetes的监控体系：

宿主机的监控数据： 比如节点的负载，CPU，内存，磁盘，网络这些常规的信息，当然你也可以查看https://github.com/prometheus/node_exporter#enabled-by-default
来看看这些指标，实在是太多了。

对apiserver，kubelet等组件的监控，比如工作队列的长度，请求的QPS和数据延迟等，主要是检查k8s本身的工作情况

k8s相关的监控数据，比如对pod，node，容器，service等主要k8s概念进行监控。

在监控指标的规划上需要遵从USE原则和RED原则

USE:

利用率
饱和度
错误率
RED原则：

4. 每秒请求数
5. 每秒错误数
6. 服务响应时间

这里需要注意： promotheus采用的是pull的模式。

138. k8s是怎么做日志监控的

一般有三种方式：

1. 将fluentd项目安装在宿主机上，然后把日志转发到远端的elsticSearch里保存起来以备检索。
   这样做的优点是： 在一个节点上只需要部署一个agent，并且不会对应用和pod有任何入侵性，这种用的比较多一些。
   缺点： 它要求应用输出日志，都必须直接输出到容器的stdout和stderr里
2. 第二种方案：当容器日志只能输出某些文件的时候，我们可以通过一个sidecar容器把这些日志文件，重新输出到sidecar的stdout和stderr上，然后在使用第一种方案。

其实第二种方案就是对第一种方案缺点的补充

3. 第三种方案，通过一个sidecar的容器，直接把应用的日志发送到远程存储里面。

这种其实也是第一种的延伸，就是把fluentd部署到pod中，后端存储还是可以用elasticsearch，知识fluentd输入源变成了应用文件日志。
但是这种方法sidecar容器会消耗过多资源。

日过日志量特比大，我们可以增加配额： 给容器上挂存储，讲日志输出到存储上。

你在回答这个问题的时候，可以说第一种方式，只要你们的日志量不大即可，如果大的话，需要加存储。

139. 【运维面试】k8s中service和ingress的区别

serivce是如何被设计的：
在pod中运行的容器在动态，弹性的变化(比如容器的重启IP地址会变化），为了给pod提供一个固定的，统一访问的接口，以及负载均衡的能力，并借助DNS系统实现服务发现功能，解决客户端发现容器难的问题，于是变设计了service

service 和pod对象的IP地址，在集群内部可达，但集群外部用户无法接入服务，解决的思路有：

1. node pod端口上做端口暴露
2. 在工作节点上用公用网络名称空间（hostname）
3. 使用service的nodeport或者loadbalancer
4. ingress七层负载和反向代理资源。

service 提供pod的负载均衡的能力，但只在4层有负载，而没有功能，只能到IP层面。

service的几种类型：

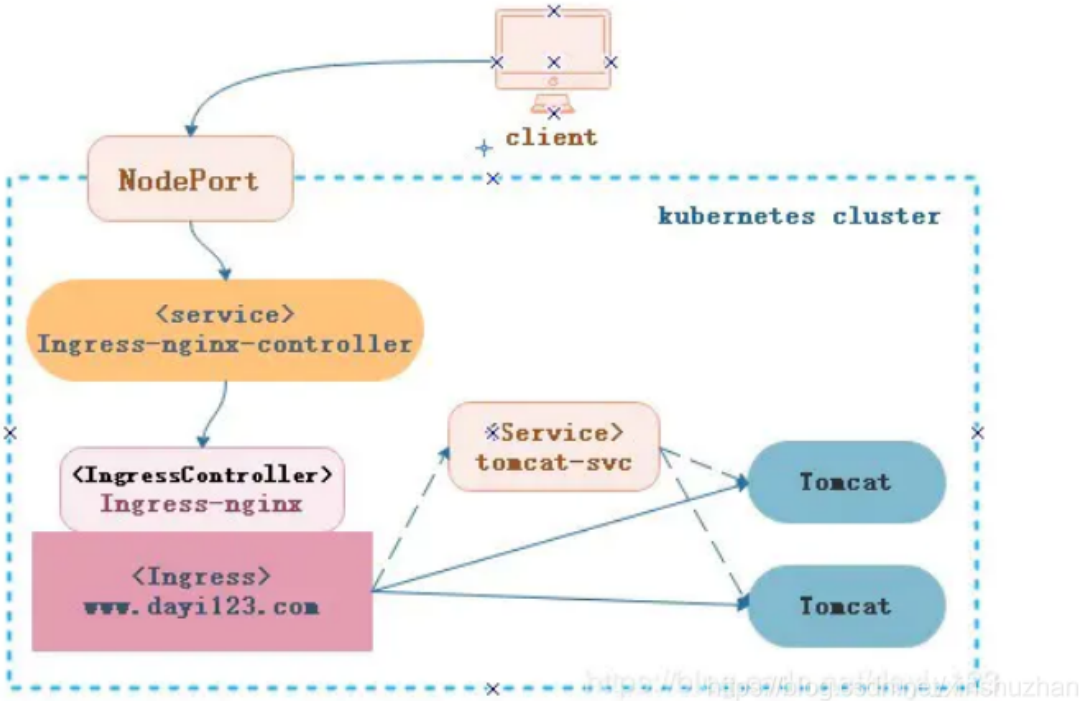
- clusetr IP： 默认类型，自动分配一个仅可以在内部访问的虚拟IP，仅供内部访问
- nodeport： 在clusterip的基础上，为集群内的每台物理机绑定一个端口，外网通过任意节点的物理机IP来访问服务，应用方式： 外服访问服务
- loadbalance: 在nodeport的基础上，提供外部负载均衡与外网统一IP，此IP可以将讲求转发到对应的服务上。 应用方式： 外服访问服务
- externalname : 引用集群外的服务，可以在集群内部通过别名的方式访问。

ingress:
service 只能提供四层的负载，虽然可以通过nodeport的方式来服务，但随着服务的增多，会在物理机上开辟太多的端口，不便于管理，所以ingress换了思路来暴露服务

创建一个具有n个副本的nginx服务，在nginx服务内配置各个服务的域名与集群内部的服务的ip，这些nginx服务在通过nodeport的方式来暴露，外部服务可以通过域名：nginx nodeport端口来访问nginx，nignx在通过域名反向代理到真实的服务器。

ingress分为ingress controller和ingress 配置。

ingress controller就是反向代理服务器，对外通过nodeport 来暴露端口。



总结：
service： 主要用来解决pod动态变化时候的IP变化，一旦podip变化，客户端就无法找到，service就是来解决这个问题的。 一个service就可以看做一组提供相同服务的pod的对外接口。
service服务于哪些pod是通过标签选择器来定义的。

而且service只能通过四层负载就是ip+端口的形式来暴露

ingress可以提供7层的负责对外暴露接口，而且可以调度不同的业务域，不同的url访问路径的业务流量。

140. k8s组件的梳理

Master组件
Master 组件对集群进行全局决策（例如，调度），并检测和响应集群事件（例如，当不满足部署的 replicas 字段时，启动新的 pod）。

1、kube-apiserver
master节点上提供k8sapi服务的组件，

2、etcd
保存了k8s集群的一些数据，比如pod的副本数，pod的期望状态与现在的状态

3、scheduler
master节点上的调度器，负责选择节点让pod在节点上运行

kube-scheduler 给一个 pod 做调度选择包含两个步骤：
1、过滤
2、打分

过滤阶段会将所有满足 Pod 调度需求的 Node 选出来。例如，PodFitsResources 过滤函数会检查候选 Node 的可用资源能否满足 Pod 的资源请求。在过滤之后，得出一个 Node 列表，里面包含了所有可调度节点；通常情况下，这个 Node 列表包含不止一个 Node。如果这个列表是空的，代表这个 Pod 不可调度。

在打分阶段，调度器会为 Pod 从所有可调度节点中选取一个最合适的 Node。根据当前启用的打分规则，调度器会给每一个可调度节点进行打分。

最后，kube-scheduler 会将 Pod 调度到得分最高的 Node 上。如果存在多个得分最高的 Node，kube-scheduler 会从中随机选取一个。

4、controller
master节点的控制器，负责在节点出现故障时进行通知和响应，负责对节点的pod状态进行监控

Node组件
1、kubelet
一个在集群中每个节点上运行的代理。它保证容器都运行在 Pod 中。
他负责管理在该节点上的属于k8s集群的容器

2、kube-proxy
一个代理，可以通过代理创建一个虚拟ip，通过这个ip来与pod进行交流

3、Container Runtime
容器运行环境是负责在节点上运行容器的软件

附加组件
1、DNS
负责对k8s集群进行域名解析

2、Dashboard
Dashboard是k8s集群的一个web界面，

3、集群层面日志
集群层面日志机制负责将容器的日志数据保存到一个集中的日志存储中，该存储能够提供搜索和浏览接口。

4、容器资源监控
容器资源监控将关于容器的一些常见的时间序列度量值保存到一个集中的数据库中，并提供用于浏览这些数据的界面。

k8s流程
供参考：

1、准备好对应的yaml文件，通过kubectl发送到Api Server中
2、Api Server接收到客户端的请求将请求内容保存到etcd中
3、Scheduler会监测etcd，发现没有分配节点的pod对象通过过滤和打分筛选出最适合的节点运行pod
4、节点会通过conteiner runntime 运行对应pod的容器以及创建对应的副本数
5、节点上的kubelet会对自己节点上的容器进行管理
6、controler会监测集群中的每个节点，发现期望状态和实际状态不符合的话，就会通知对应的节点
7、节点收到通知，会通过container runtime来对pod内的容器进行收缩或者扩张

常见问题：

K8S是如何对容器编排?
在K8S集群中，容器并非最小的单位，K8S集群中最小的调度单位是Pod，容器则被封装在Pod之中。由此可知，一个容器或多个容器可以同属于在一个Pod之中。

Pod是怎么创建出来的?
Pod是由Pod控制器进行管理控制，其代表性的Pod控制器有Deployment、StatefulSet等。
（1）客户端提交创建请求，可以通过API Server的Restful API，也可以使用kubectl命令行工具。支持的数据类型包括JSON和YAML。
（2）API Server处理用户请求，存储Pod数据到etcd。
（3）调度器通过API Server查看未绑定的Pod。尝试为Pod分配主机。
（4）过滤主机 (调度预选)：调度器用一组规则过滤掉不符合要求的主机。比如Pod指定了所需要的资源

量，那么可用资源比Pod需要的资源量少的主机会被过滤掉。
（5）主机打分(调度优选)：对第一步筛选出的符合要求的主机进行打分，在主机打分阶段，调度器会考虑一些整体优化策略，比如把容一个Replication Controller的副本分布到不同的主机上，使用最低负载的主机等。
（6）选择主机：选择打分最高的主机，进行binding操作，结果存储到etcd中。
（7）kubelet根据调度结果执行Pod创建操作： 绑定成功后，scheduler会调用APIServer的API在etcd中创建一个boundpod对象，描述在一个工作节点上绑定运行的所有pod信息。运行在每个工作节点上的kubelet也会定期与etcd同步boundpod信息，一旦发现应该在该工作节点上运行的boundpod对象没有更新，则调用Docker API创建并启动pod内的容器。

Pod资源组成的应用如何提供外部访问的?
Pod组成的应用是通过Service这类抽象资源提供内部和外部访问的，但是service的外部访问需要端口的映射，带来的是端口映射的麻烦和操作的繁琐。为此还有一种提供外部访问的资源叫做Ingress。

Service又是怎么关联到Pod呢?
在上面说的Pod是由Pod控制器进行管理控制，对Pod资源对象的期望状态进行自动管理。而在Pod控制器是通过一个YAML的文件进行定义Pod资源对象的。在该文件中，还会对Pod资源对象进行打标签，用于Pod的辨识，而Servcie就是通过标签选择器，关联至同一标签类型的Pod资源对象。这样就实现了从service–>pod–>container的一个过程。

可以这么理解： pod是一个虚拟机，容器就是一个应用程序，k8s相当于操作系统，镜像相当于.exe安装包。

141. 关于tcp/IP协议

OSI七层模型协议

OSI的七层协议主要包括:

- 物理层（physical layer）
- 数据链路层（data link layer）
- 网络层（network layer）
- 运输层（transport layer）
- 会话层（session layer）
- 表示层（presentation layer）
- 应用层（application layer）

每一层的协议

- 物理层：RJ45、CLOCK、IEEE802.3（中继器、集线器）,ISO2110,光导纤维，双绞线
- 数据链路层：wi-fi, ATM, DTM,令牌环，以太网，PPPoE ,PPP、FR、HDLC、VLAN、MAC（网桥、交换机）
- 网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP（交换机）
- 传输层：TCP（T/TCP · Fast Open） UDP DCCP SCTP RSVP PPTP TLS/SSL
- 会话层：NFS、SQL、NETBIOS、RPC
- 表示层：JPEG、MPEG、ASII
- 应用层：DHCP（v6） DNS FTP Gopher HTTP（SPDY、HTTP/2） IMAP4 IRC NNTP XMPP POP3 SIP SMTP SNMP SSH TELNET RPC RTCP RTP RTSP SDP SOAP GTP STUN NTP SSDP

故事版OSI七层模型

一封邮件的自白:

我是一封邮件， 目的地是广州的一个地址，我来到应用层，找到了SMTP邮件协议，来到表示层的进行处理。
表示层要把主人写好的邮件，进行编码和转换，变成我们传输通道里的伙伴们都认识的一种形式。

等待表示层把数据处理完后，其实就已经把数据都知准备好了，这时候我会联系我的代言人会话层进行与主机通话，相当于打电话那样的，目的是把我的主机和服务器建立连接，方便我把数据传输过去。

主机： 你好，我是主机，请问你是邮件服务器吗？，我需要和你建立连接。
服务器：你好，我是邮件服务器，连接已经建立。

建立连接后，就开始传输数据了，传输层的任务比较艰巨，他不但要传输数据，还要处理在传输过程中可能出现的异常，比如数据丢包。

在传输的过程中，数据就开始顺着网络层指定的路线进行传输了， 网络错综负责，稍有不慎就走错路，所以网络层在这里就起到了规划路线的作用，相当于快递公司，告诉数据从哪里走最近。

数据到了广州后，就到了数据链路层的手里，数据链路层相当于快递员， 快递员可以精准的找到每个本地网络中的设备，然后把数据精准到传送到相应的设备上去。

剩下的就是物理层了，物理层主要做信号转换和物理传输。

相关面试题

1. TCP和UDP的 区别：

答：

```
TCP是面向有连接型，UDP是面向无连接型；
TCP是一对一传输，UDP支持一对一、一对多、多对一和多对多的交互通信；
TCP是面向字节流的，即把应用层传来的报文看成字节流，将字节流拆分成大小不等的数据块，并添加TCP首部；UDP是面向报文的，对应用层传下来的报文不拆分也不合并，仅添加UDP首部；
TCP支持传输可靠性的多种措施，包括保证包的传输顺序、重发机制、流量控制和拥塞控制；UDP仅提供最基本的数据传输能力。
```

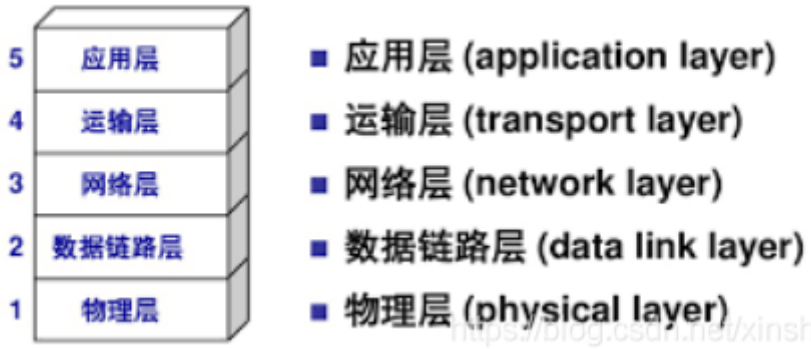
2. TCP对应的应用层协议有哪些？UDP对应的应用层协议有哪些？

```
TCP对应的典型的应用层协议：

FTP：文件传输协议；
SSH：远程登录协议；
HTTP：web服务器传输超文本到本地浏览器的超文本传输协议。
UDP对应的典型的应用层协议：

DNS：域名解析协议；
TFTP：简单文件传输协议；
SNMP：简单网络管理协议。
```

3. 常见的http动词有哪些？

```
GET：从服务器获取资源
POST：在服务器新建资源
PUT：在服务器更新资源
DELETE：在服务器删除资源
HEAD：获取资源的元数据
OPTIONAL：查询对指定的资源支持的方法
```

4. 五层协议的体系结构，都有哪些协议？



5. ping命令基于哪一层协议的原理是什么?

ping命令是基于网络层的命令，是基于ICMP协议工作的。

142. 谈谈你对CDN的理解

在淘宝的图片访问中，有98%的流量走了CDN缓存，只有2%会回源到源站，节省了大量的服务器资源。

但如果遇到高峰期，图片内容大批量发生变化，大量用户访问会穿透CDN，对源站造成压力。

cdn的工作原理

内容分发网络（cdn）是建立并覆盖在承载网上，由分布在不同区域的边缘节点服务器群组成的分布式网络。

CDN应用广泛，比如图片小文件，大文件下载，视音频点播，直播流媒体，全站加速，安全加速等。

- 当用户访问，www.geekyunwei.com(我的一个个人运维网站）时，首先向本地DNS发起域名解析。
- 如果本地DNS缓存中有www.geekyunwei.com的IP地址记录，则给用户返回。 如果没有则像授权DNS查询。
- 当授权DNS解析www.geekyunwei.com时，返回域名对应的IP地址。
- 域名解析请求发送到DNS调度系统，并为请求分配最佳节点的IP地址。
- 本地DNS获取DNS返回的解析IP地址
- 用户获取解析的IP地址
- 然后用户像获取的IP地址发起资源请求

注意： CDN的加速资源是跟域名绑定的
CDN的意图就是尽可能的减少资源在转发、传输、链路抖动等情况下顺利保障信息的连贯性

一个简单的CDN网络是由一个DNS服务器和几台缓存服务器组成。

其他问题:

1. CDN加速是对网站所在服务器加速，还是对其域名加速?

CDN是只对网站的某一个具体的域名加速。如果同一个网站有多个域名，则访客访问加入CDN的域名获得加速效果，访问未加入CDN的域名，或者直接访问IP地址，则无法获得CDN效果。

2. CDN和镜像站点比有何优势

CDN对网站的访客完全透明，不需要访客手动选择要访问的镜像站点，保证了网站对访客的友好性。CDN对每个节点都有可用性检查，不合格的节点会第一时间剔出，从而保证了极高的可用率，而镜像站点无法实现这一点。CDN部署简单，对原站基本不做任何改动即可生效。

3. 为什么我的网站更新后，通过CDN后看到网页还是旧网页，如何解决?

由于CDN采用各节点缓存的机制，网站的静态网页和图片修改后，如果CDN缓存没有做相应更新，则看到的还是旧的网页。为了解决这个问题，CDN管理面板中提供了URL推送服务，来通知CDN各节点刷新自己的缓存。在URL推送地址栏中，输入具体的网址或者图片地址，则各节点中的缓存内容即被统一删除，并且当即生效。如果需要推送的网址和图片太多，可以选择目录推送，输入 http://www.kkk.com/news 即可以对网站下news目录下所有网页和图片进行了刷新

4. 哪些情况下推荐使用CDN
   一般来说以资讯、内容等为主的网站，具有一定访问体量的网站资讯网站、政府机构网站、行业平台网站、商城等以动态内容为主的网站论坛、博客、交友、SNS、网络游戏、搜索/查询、金融等。提供http下载的网站软件开发商、内容服务提供商、网络游戏运行商、源码下载等有大量流媒体点播应用的网站拥有视频点播平台的电信运营商、内容服务提供商、体育频道、宽频频道、在线教育、视频博客等

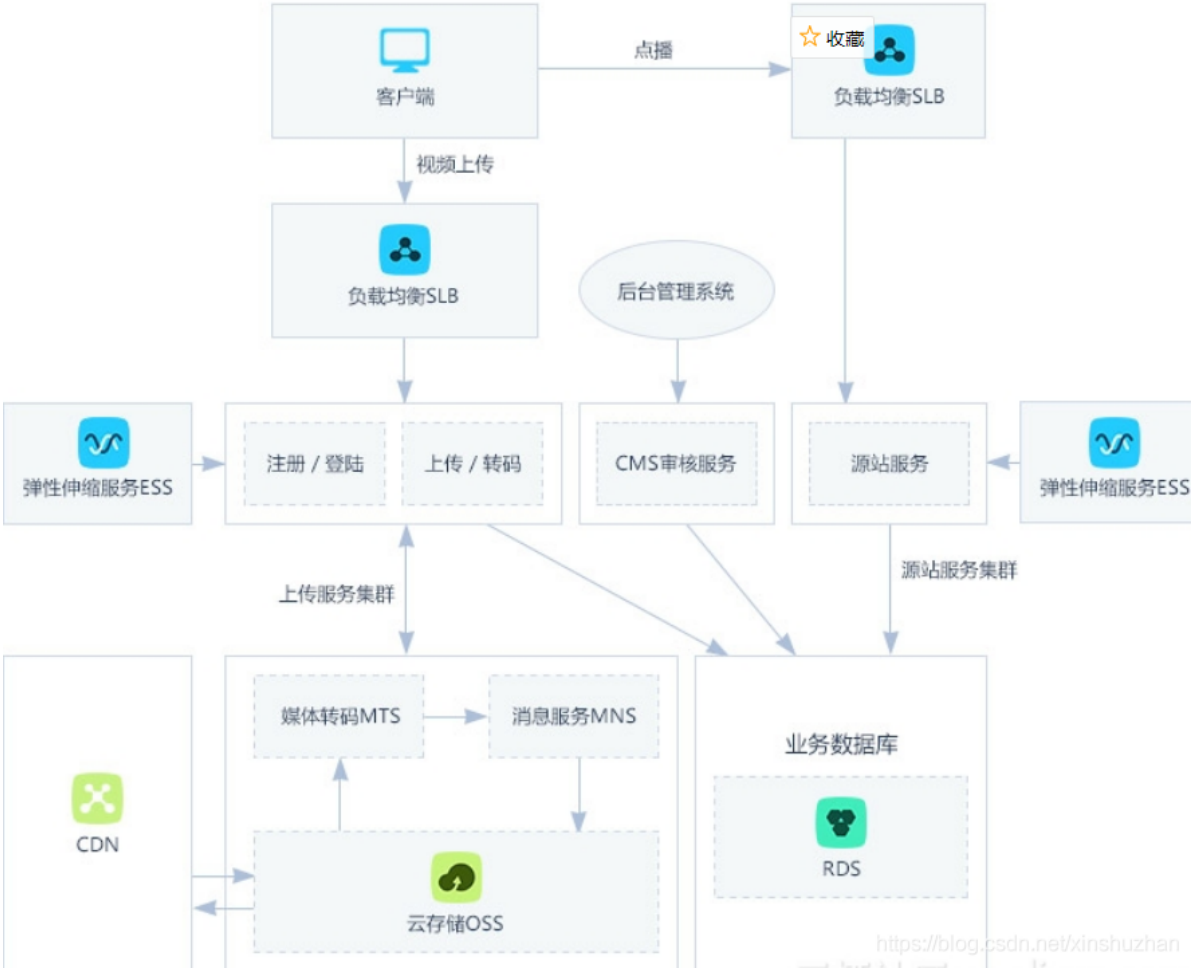
CDN的主要场景：

1. 网站加速
2. 大文件下载加速
3. 音视频加速
   其他功能： 全站加速，安全加速

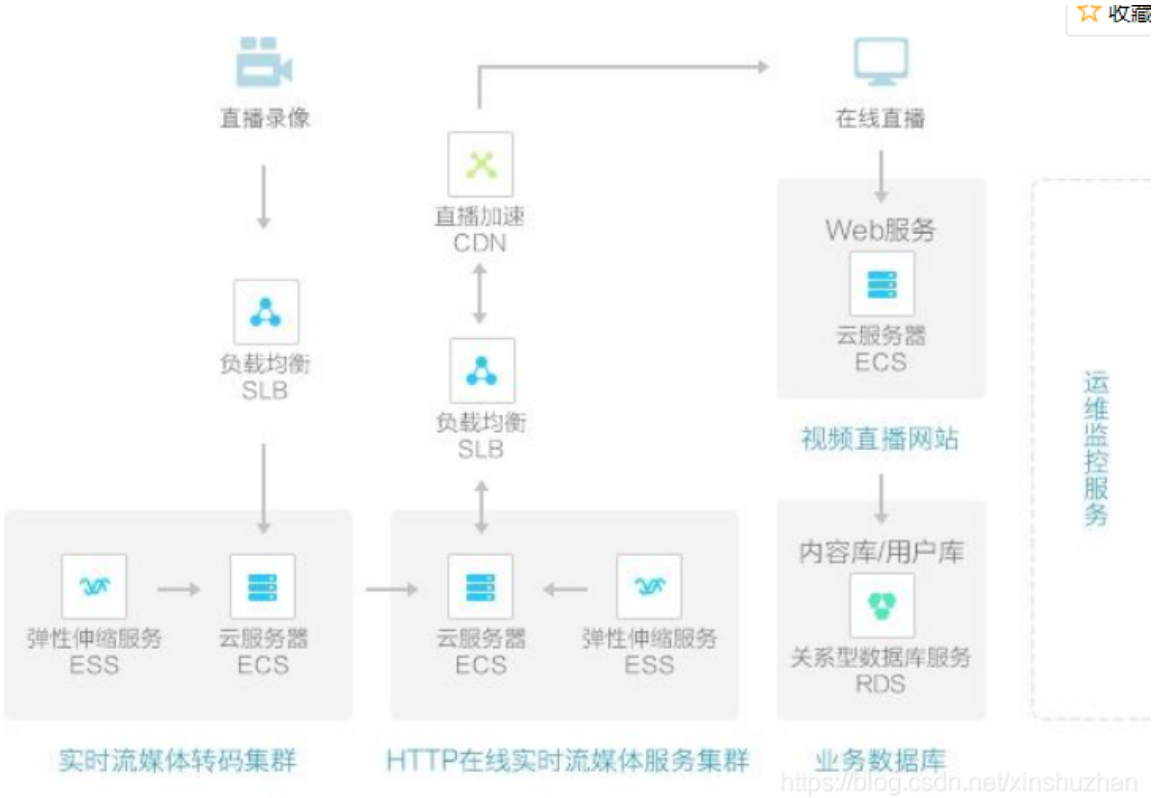
推荐书籍： 内容分发网络（cdn）原理与实践

架构图

*视频音频点播及大文件分发下载架构图：**



视频直播加速：



国内有哪些比较好的CDN厂商

- 蓝汛
- 网宿
- 阿里云
- 腾讯云
- upyun
- ucloud
- 七牛云